

AD-A050 967

LOGICON INC SAN DIEGO CALIF

F/G 9/2

TERMINAL ACCESS SYSTEM (TAS) SYSTEM DESIGN SPECIFICATION.(U)

JAN 78 R M BARNHART, L R ERICKSON

N00014-76-C-0899

76-C-0899-6

NL

UNCLASSIFIED

1 OF 8

AD
A050 967



AD A 050967

DDC FILE COPY

Report 76-C-0899-6

TERMINAL ACCESS SYSTEM (TAS)
SYSTEM DESIGN SPECIFICATION

Logicon, Inc.
4010 Sorrento Valley Blvd., P. O. Box 80158
San Diego, California 92138

30 January 1978

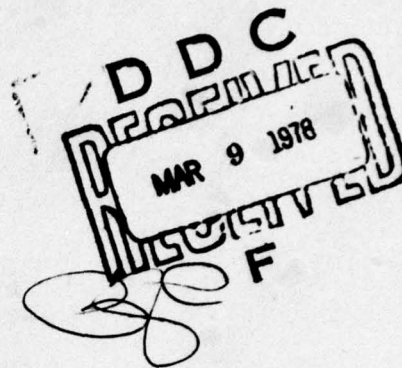
Final Report for Period 1 November 1977 - 30 January 1978

DOD DISTRIBUTION STATEMENT

Approved for public release;
distribution unlimited.

DEFENSE ADVANCED RESEARCH PROJECTS AGENCY
1400 Wilson Boulevard
Arlington, VA 22209

OFFICE OF NAVAL RESEARCH
INFORMATION SYSTEMS PROGRAM
Code 437
Arlington, VA 22217



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 76-C-899-6	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Terminal Access System (TAS) System Design Specification	5. TYPE OF REPORT & PERIOD COVERED Final Report 1 Nov 77-30 Jan 78	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) R. M. Barnhart, L. R. Erickson, M. E. Soglead S. L. Westermarck	8. CONTRACT OR GRANT NUMBER(s) N00014-76-C-899	9. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
9. PERFORMING ORGANIZATION NAME AND ADDRESS Logicon, Inc. 4010 Sorrento Valley Blvd., P. O. Box 80158 San Diego, California 92138	10. REPORT DATE 30 Jan 78	11. NUMBER OF PAGES 677 P.
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209	12. SECURITY CLASS. (of this report) Unclassified	13. DECLASSIFICATION/DOWNGRADING SCHEDULE
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	15. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.	
16. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
17. SUPPLEMENTARY NOTES		
18. KEY WORDS (Continue on reverse side if necessary and identify by block number) ARPANET Front-End Processing Data Base Management Systems (DBMS) UNIX Multi-Level Security Networking		
19. ABSTRACT (Continue on reverse side if necessary and identify by block number) The Terminal Access System (TAS) provides a uniform environment for computer naive intelligence analysts as they query data bases distributed on an ARPA-like network. TAS resides on a PDP-11/70 processor and operates under the UNIX Operating System. TAS performs as a user host on the network. TAS is composed of a large set of application processes which directly satisfy user entered commands involving text file editing.		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

406012

set

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

20. Abstract

system status reports, and display of query responses. In addition to the TAS application processes, there is a set of system processes which provide fundamental interfaces for networking, access authorization, and network logging. These processes are designed such that other subsystems destined for TAS occupancy can utilize their facilities. The ARPA Data Access and Presentation Terminal (ADAPT) system is one such system. Key TAS system processes are the Batch Query and Response Dispatcher (BQRD) which provides internal management for all batch queries, the Intelligence Network Interface (INI) process, which emulates the COINS I protocol for TAS, the Interactive Query Interface (IQI) process, which provides system interfaces for interactive network transactions, the Access Authorization Process (AAP) which provides complete control of user/terminal access to the hosts and files residing on the COINS II network, and the TAS Logging Process (TLP) conglomerate which provides interfaces for network logging, both batch and interactive, and access authorization violations. Although most of TAS/user interfaces are controlled by a specially modified UNIX shell process, the sophisticated Access Authorization File Maintenance (AAFM) subsystem residing under TAS utilizes a front-end generated by the Yet Another Compiler-Compiler (YACC) for user interfaces. The entire TAS operates over an elaborate UNIX hierarchical file system composed of directories and structured data files.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

TABLE OF CONTENTS

	<u>Page</u>
INTRODUCTION	13
TAS ENVIRONMENT	16
TAS File Environment	16
TAS Process Flow and Communication	29
TAS PROCESS DESCRIPTIONS	31
AAFM	35
Global Data Usage	35
Local Data Usage	36
General Process Flow	37
Major Function Descriptions	40
AALOCK	88
Global Data Usage	88
Local Data Usage	88
General Process Flow	88
AALISTSNI	90
Global Data Usage	90
Local Data Usage	91
General Process Flow	91
Major Function Descriptions	92
AARMVSNi	102
Global Data Usage	102
Local Data Usage	102
General Process Flow	102
Major Function Descriptions	103
AATERM	112
Global Data Usage	112
Local Data Usage	113
General Process Flow	114
Major Function Descriptions	116
AAP	146
Global Data Usage	146
Local Data Usage	147
General Process Flow	147
Major Function Descriptions	148
AUTH	154
Global Data Usage	154
Local Data Usage	155

ACCESSION for	
NTIS	White Section <input checked="" type="checkbox"/>
DDC	Buff Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUST 10A-1-1	
BY	
DISTRIBUTION/ABIT/CS	
Dist.	SIAL
A	

TABLE OF CONTENTS (Cont)

	<u>Page</u>
General Process Flow	155
Major Function Descriptions	156
BQP	161
Global Data Usage	161
Local Data Usage	161
General Process Flow	162
Major Function Descriptions	163
BQRD	182
Global Data Usage	182
Local Data Usage	182
General Process Flow	182
Major Function Descriptions	188
CANCEL	214
Global Data Usage	214
Local Data Usage	214
General Process Flow	214
Major Function Descriptions	215
CLOSE	221
Global Data Usage	221
Local Data Usage	221
General Process Flow	221
Major Function Descriptions	221
DATEIS	223
Global Data Usage	223
Local Data Usage	223
General Process Flow	223
DISPLAY/PRINT	225
Global Data Usage	225
Local Data Usage	225
General Process Flow	225
ECHO	230
Global Data Usage	230
Local Data Usage	230
General Process Flow	230
EDIT	232
Global Data Usage	232
Local Data Usage	232
General Process Flow	232

TABLE OF CONTENTS (Cont)

	<u>Page</u>
FILIST	241
Global Data Usage	241
Local Data Usage	241
General Process Flow	241
Major Function Descriptions	242
HELP	247
Global Data Usage	247
Local Data Usage	247
General Process Flow	247
HOLD	249
Global Data Usage	249
Local Data Usage	249
General Process Flow	249
INI	251
Global Data Usage	251
Local Data Usage	251
General Process Flow	251
Major Function Descriptions	256
IQI	278
Global Data Usage	278
Local Data Usage	278
General Process Flow	278
Major Function Descriptions	279
IQIR	293
Global Data Usage	293
Local Data Usage	293
General Process Flow	293
JOBID	295
Global Data Usage	295
Local Data Usage	295
General Process Flow	295
Major Function Descriptions	295
JOBSTAT	298
Global Data Usage	298
Local Data Usage	298
General Process Flow	298
Major Function Descriptions	299
LIST	303
Global Data Usage	303

TABLE OF CONTENTS (Cont)

	<u>Page</u>
Local Data Usage	303
General Process Flow	303
LOGOFF	305
Global Data Usage	305
Local Data Usage	305
General Process Flow	305
Major Function Descriptions	306
LOGON	310
Global Data Usage	310
Local Data Usage	310
General Process Flow	310
Major Function Descriptions	313
LOGTAPE	319
Global Data Usage	319
Local Data Usage	319
General Process Flow	319
Major Function Descriptions	319
LTD	322
Global Data Usage	322
Local Data Usage	322
General Process Flow	322
Major Function Descriptions	323
MSG	328
Global Data Usage	328
Local Data Usage	328
General Process Flow	328
Major Function Descriptions	329
PROJECT	333
Global Data Usage	333
Local Data Usage	333
General Process Flow	333
PURGE	336
Global Data Usage	336
Local Data Usage	336
General Process Flow	336
Major Function Descriptions	337
RD	342
Global Data Usage	342
Local Data Usage	342

TABLE OF CONTENTS (Cont)

	<u>Page</u>
General Process Flow	342
Major Function Descriptions	344
RECLAIM	350
Global Data Usage	350
Local Data Usage	350
General Process Flow	350
Major Function Descriptions	352
RECOVER	358
Global Data Usage	358
Local Data Usage	358
General Process Flow	358
TRCC	361
Global Data Usage	361
Local Data Usage	361
General Process Flow	361
Major Function Descriptions	362
TRCC2	382
Global Data Usage	382
Local Data Usage	382
General Process Flow	383
Major Function Descriptions	383
REMOVE	405
Global Data Usage	405
Local Data Usage	405
General Process Flow	405
RESEND	410
Global Data Usage	410
Local Data Usage	410
General Process Flow	410
RESPONSE	414
Global Data Usage	414
Local Data Usage	414
General Process Flow	414
Major Function Descriptions	414
SAVE	417
Global Data Usage	417
Local Data Usage	417
General Process Flow	417

TABLE OF CONTENTS (Cont)

	<u>Page</u>
SHOW	421
Global Data Usage	421
Local Data Usage	421
General Process Flow	421
SOLIS	424
Global Data Usage	424
Local Data Usage	424
General Process Flow	424
Major Function Descriptions	425
SOLISW	433
Global Data Usage	433
Local Data Usage	433
General Process Flow	433
SPOOL	436
Global Data Usage	436
Local Data Usage	436
General Process Flow	436
SPURGE	438
Global Data Usage	438
Local Data Usage	438
General Process Flow	438
SRESP	442
Global Data Usage	442
Local Data Usage	442
General Process Flow	442
Major Function Descriptions	443
SRVC	447
Global Data Usage	447
Local Data Usage	447
General Process Flow	447
STATUS	454
Global Data Usage	454
Local Data Usage	454
General Process Flow	454
Major Function Descriptions	454
SYSRES	457
Global Data Usage	457
Local Data Usage	457
General Process Flow	457
Major Function Descriptions	458

TABLE OF CONTENTS (Cont)

	<u>Page</u>
SYSTAT	463
Global Data Usage	463
Local Data Usage	463
General Process Flow	463
Major Function Descriptions	464
TALEND	468
Global Data Usage	468
Local Data Usage	468
General Process Flow	468
TALIST	470
Global Data Usage	470
Local Data Usage	470
General Process Flow	470
Major Function Descriptions	471
TASDOWN	492
Global Data Usage	492
Local Data Usage	492
General Process Flow	492
TASDUMP	498
Global Data Usage	498
Local Data Usage	498
General Process Flow	498
TASLOAD	500
Global Data Usage	500
Local Data Usage	500
General Process Flow	500
TD	503
Global Data Usage	503
Local Data Usage	503
General Process Flow	503
TINIT	505
Global Data Usage	505
Local Data Usage	505
General Process Flow	505
Major Function Descriptions	506
TLP	509
Global Data Usage	509
Local Data Usage	509
General Process Flow	510
Major Function Descriptions	510

TABLE OF CONTENTS (Cont)

	<u>Page</u>
TRACE	538
Global Data Usage	538
Local Data Usage	538
General Process Flow	538
Major Function Descriptions	538
TRACK	542
Global Data Usage	542
Local Data Usage	542
General Process Flow	542
Major Function Descriptions	543
TSUM	546
Global Data Usage	546
Local Data Usage	546
General Process Flow	546
Major Function Descriptions	547
USTAT	550
Global Data Usage	550
Local Data Usage	550
General Process Flow	550
WD	552
Global Data Usage	552
Local Data Usage	552
General Frocess Flow	552
Major Function Descriptions	553
TAS GLOBAL FUNCTIONS AND DATA	557
Global Functions	557
GFERROR	558
GFLOCK	559
GFUNLOCK	559
GFJOBID	560
GFMOVE	561
GFNEQUAL	561
TAS Global Data	562
GACCM	564
GAMAP	564
GAPNM	564
GAPPL	564
GAUTH	565

TABLE OF CONTENTS (Cont)

	<u>Page</u>
GBQFN	567
GBQST	568
GCLAS	569
GDOWN	570
GFEDES	570
GFEMES	571
GFILE	571
GFILS	572
GHOST	573
GILOG	574
GIMSG	574
GJOBID	576
GJOBS	576
GLBUF	578
GLJOBS	578
GMAP	580
GMASK	581
GMESG	583
GOPARAMS	584
GPIDS	585
GPROJ	586
GQFIL	587
GRESF	588
GTAUT	588
GTCTS	590
GTERM	592
GUIDES	593
GUTY	594
LTPID	594
LTSIZE	595
TLTAL	595
APPENDIX A - ACCESS AUTHORIZATION FILE MAINTENANCE LANGUAGE	614
APPENDIX B - TAS COMMAND SUMMARY AND SYNTAX	642
APPENDIX C - TAS SYSTEM PROCESS INTERFACES	649

LIST OF ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
1	TAS/UNIX Root Directory Entries	17
2	TAS System Directory Entries	18
3	TAS Help Directory Entries.	20
4	TAS Host Directory Entries.	22
5	TAS Logging Directory Entries	24
6	TAS User Directory Entries.	26
7	TAS Process Flow and Communication	30
8	AAFM Process Data Flow	56
9	AALOCK Process Data Flow	89
10	AALIST SNI Process Data Flow	96
11	AARMVSNi Process Data Flow.	106
12	AATERM Process Data Flow	127
13	AAP Process Data Flow	150
14	AUTH Process Data Flow	158
15	BQP Process Data Flow	167
16	BQRD Process Data Flow	192
17	CANCEL Process Data Flow	217
18	CLOSE Process Data Flow	222
19	DATEIS Process Data Flow.	224
20	DISPLAY Process Data Flow	227
21	ECHO Process Data Flow	321
22	EDIT Process Data Flow.	324
23	FILIST Process Data Flow	244
24	HELP Process Data Flow	248
25	HOLD Process Data Flow	250
26	INI Process Data Flow	260
27	IQI Process Data Flow	285
28	IQIR Process Data Flow	294
29	JOBID Process Data Flow	296
30	JOBSTAT Process Data Flow	300
31	LIST Process Data Flow.	304
32	LOGOFF Process Data Flow	307
33	LOGON Process Data Flow	314
34	LOGTAPE Process Data Flow	321
35	LTD Process Data Flow	325
36	MSG Process Data Flow	330
37	PROJECT Process Data Flow	335
38	PURGE Process Data Flow	338
39	RD Process Data Flow	346

LIST OF ILLUSTRATIONS (Cont)

<u>Figure</u>		<u>Page</u>
40	RECLAIM Process Data Flow	354
41	RECOVER Process Data Flow	360
42	TRCC Process Data Flow	369
43	TRCC2 Process Data Flow	389
44	REMOVE Process Data Flow	408
45	RESEND Process Data Flow	412
46	RESPONSE Process Data Flow	415
47	SAVE Process Data Flow	419
48	SHOW Process Data Flow	422
49	SOLID Process Data Flow	428
50	SOLISW Process Data Flow	434
51	SPOOL Process Data Flow	437
52	SPURGE Process Data Flow	440
53	SRESP Process Data Flow	444
54	SRVC Process Data Flow	450
55	STATUS Process Data Flow	455
56	SYSRES Process Data Flow	459
57	SYSTAT Process Data Flow	465
58	TALEND Process Data Flow	469
59	TALIST Process Data Flow	474
60	TASDOWN Process Data Flow	495
61	TASDUMP Process Data Flow	499
62	TASLOAD Process Data Flow	502
63	TD Process Data Flow	504
64	TINIT Process Data Flow	507
65	TLP Process Data Flow	518
66	TRACE Process Data Flow	540
67	TRACK Process Data Flow	544
68	TSUM Process Data Flow	548
69	USTAT Process Data Flow	551
70	WD Process Data Flow	554
71	GAPPL Item Description	565
72	GAUTH Item Description	566
73	GBQFN Item Description	568
74	GBQST Item Description	568
75	Classification Codes	570
76	GFEDES Item Description	571
77	GFILE Item Description	572
78	GFLIS Item Description	573

LIST OF ILLUSTRATION (Cont)

Figure		Page
79	GHOST Item Description	574
80	GIMSG Item Description	575
81	GJOBS Item Description	577
82	GLBUF Description	579
83	CLJOBS Item Description	580
84	Classification Bit Maps	581
85	Determination of Access Authorization	583
86	GMESG Item Description	583
87	COPARAMS Item Description	585
88	GPROJ Item Description	586
89	GQFIL Item Description	588
90	GTAUT Item Description	589
91	GTCTS Item Description	591
92	GTERM Item Description	593
93	GUDES Item Description	594
94	TLTAL Record Header Description	596
95	Description of Event 00 Record	597
96	Description of an Event 01 Record	598
97	Description of an Event 02 Record	600
98	Description of an Event 03 Record	601
99	Description of an Event 04 Record	602
100	Description of an Event 05 Record	603
101	Description of an Event 07 Record	605
102	Description of an Event 08 Record	606
103	Description of an Event 09 Record	607
104	Description of an Event 10 Record	608
105	Description of an Event 11 Record	610
106	Description of an Event 12 Record	611
107	Description of an Event 13 Record	613
A-1	AAFM Keywords	620

INTRODUCTION

The Terminal Access System (TAS) provides a uniform terminal interface to "computer naive" intelligence analysts for querying data bases distributed on the COINS II Network. TAS operates in a PDP-11/70 processor under the UNIX Operating System. The PDP-11/70 processor and its supplementary hardware suite communicate over the COINS II Network through an Interface Message Processor (IMP) by way of a Local-Host/Distant-Host (LH-DH/11) interface. TAS, in general, behaves as a "user host" on the ARPA-like COINS II Network.

TAS, which is now operational on COINS II and resides at the COINS Project Management Office, provides COINS II users with many useful functions including:

- a. uniform terminal interfaces (TTY MOD-40) for all TAS command entries,
- b. uniform networking for conventional interactive host communications and COINS I "batch" host transactions,
- c. complete query generation and maintenance facilities via text editing tools, and
- d. sophisticated multi-level security control facilities for user/terminal access authorization to hosts at the file level.

This document comprises a complete program design specification for TAS. It details the overall design approach used in TAS development, including all its interfaces with the UNIX Operating System and its Network Control Program (NCP). This design specification consists of four basic sections and three appendices. After this introductory section, the second

section presents the overall design of the TAS environment including a general overview of process communication in TAS and a detailed description of the UNIX file hierarchy used in its operation with the description of all directory/file hierarchies. This section should be read before attempting to read the individual process or data structure descriptions contained in other sections.

The third section contains a detailed description of every process in TAS. Process descriptions are placed in alphabetical order for easy reference. Each description begins with a brief statement of the purpose of the process, followed by a list of TAS global data used by this process and a list of any pertinent local data. Next is an overall description of the basic process data flow, calling out by name interactions with TAS global data, calls to important functions, and interactions with other TAS processes. All salient functions called out in the general data flow are individually described in detail. Finally, general flowcharts are provided for each named function.

The fourth section contains a complete list and description of all TAS global functions and data structures. Global functions are those functions which are used by more than one process in TAS. Global data is that data which exists as a single copy and is referenced by more than one process. Each global data structure (either a C structure or array) is described as follows. A brief statement of its purpose is followed by a list of all defined fields (if a structure) and a short description of their use. A diagram showing a graphic representation of the structure accompanies each description. A knowledge of these data structures as they fit into the TAS file/directory hierarchy is mandatory for an overall understanding of TAS's operation.

Appendix A gives the Yet Another Compiler-Compiler (YACC) language specifications utilized in TAS. The specifications include language "token" assignments and the Backus-Normal-Form (BNF) presentation of the required grammar for language specifications of the Access Authorization File Maintenance (AAFM) subsystem interfaces. Preceding the YACC specifications is the syntax of the AAFM language. Appendix B provides a short summary of the complete TAS command set. This includes the syntax plus a brief semantic description of each command. Appendix C presents required calling sequences for all TAS system processes. TAS system processes include the Batch Query and Response Dispatcher (BQRD) process, the Interactive Query Interface (IQI) process, the Access Authorization Process (AAP), and the TAS Logging Process conglomerate (TLP). This appendix pays special attention to all required calling sequences and their parameters, possible statuses that might be generated whether they are derivable from typical execute/wait sequences or from more esoteric paths such as pipes or signals.

TAS ENVIRONMENT

This section describes the TAS system as it operates in the UNIX environment. The overall design of TAS is presented in two parts: 1) a detailed description of the TAS file environment, and 2) a general overview of the communication paths exercised by the various TAS processes.

TAS FILE ENVIRONMENT - The UNIX root directory is also the root directory for the TAS hierarchical file and directory environment. This root directory, "/", contains five entries pertinent to the operation of TAS. The TAS object directory "/tas" contains the executable code files for the TAS commands; the UNIX directory "/usr" contains the TAS system directories; the UNIX directories "/u1" and "/u2" contain all TAS user directories; and lastly, the UNIX directory "/etc" contains the UNIX initialization command file, "/etc/rc", and the UNIX user password file, "/etc/passwd", both utilized by TAS processes. (See figure 1.)

The TAS object directory, "/tas", contains entries for the executable code files for the TAS commands. One of its entries points to the "/tas/sys" directory which contains entries for the executable code files for the remaining TAS system processes. The UNIX directory "/usr" contains only one entry pertinent to the TAS file and directory system. This entry is for the TAS file system directory "/usr/tsy" which has entries for six system files and four directory pointers. (See figure 2.) The six pathnames for the TAS system files are:

- | | | | |
|----|-------------------|---|----------------------------|
| 1. | "/usr/tsy/gfedes" | - | error/message descriptions |
| 2. | "/usr/tsy/gfemes" | - | error/message text |
| 3. | "/usr/tsy/gjobid" | - | JOBID counter |

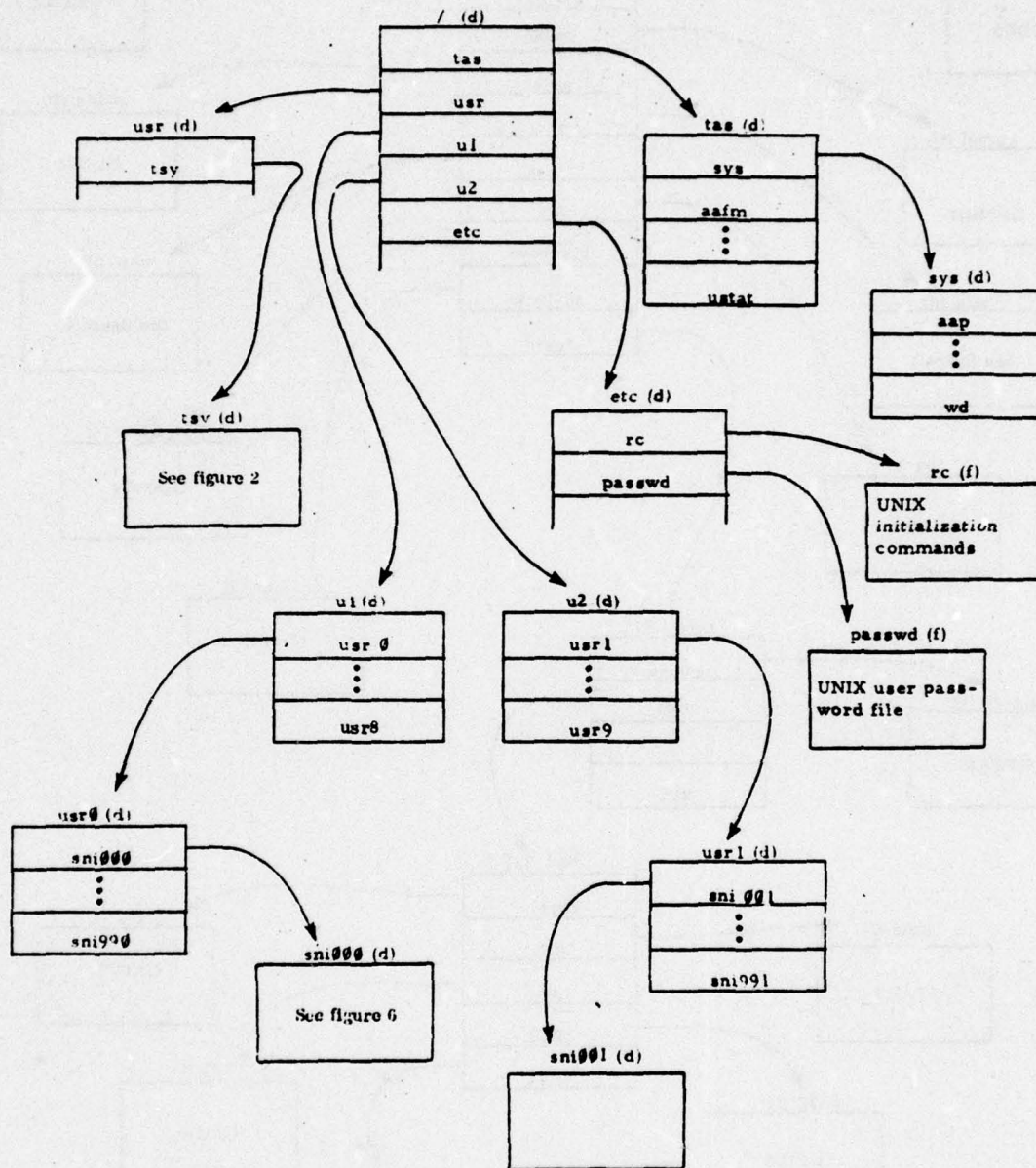


Figure 1. TAS/UNIX Root Directory Entries

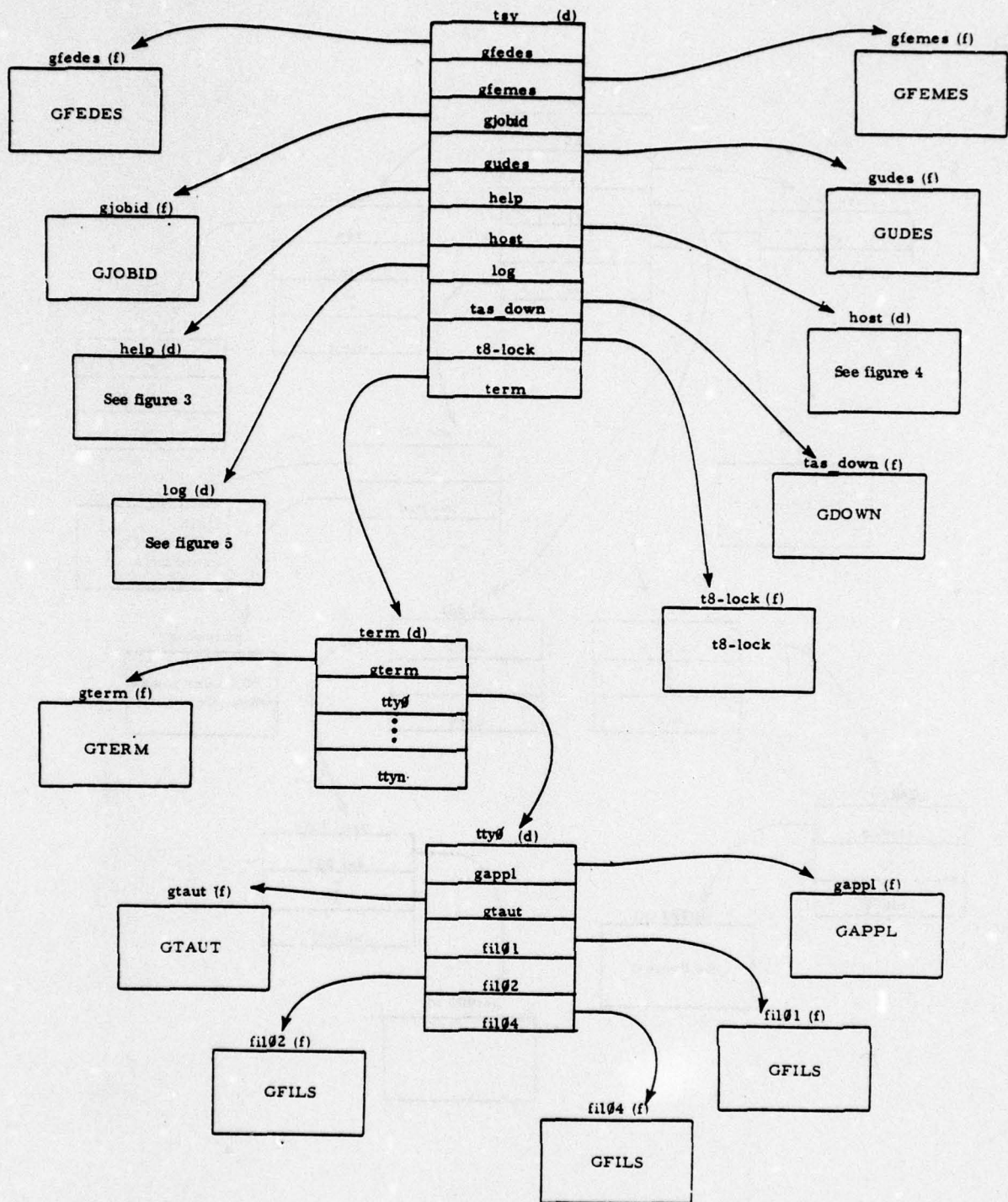


Figure 2. TAS System Directory Entries

4. `"/usr/tsy/gudes"` - user descriptions
5. `"/usr/tsy/tas_down"` - TASDOWN's temporary lock file
6. `"/usr/tsy/t8-lock"` - temporary operator's console lock file.

The four permanent directory entries in `"/usr/tsy"` are:

1. `"/usr/tsy/help"` - help directory
2. `"/usr/tsy/host"` - host directory
3. `"/usr/tsy/log"` - logging directory
4. `"/usr/tsy/term"` - terminal directory

The help directory `"/usr/tsy/help"` contains files which have information describing the TAS commands. The general help file (pathname `"/usr/tsy/help/all"`) contains the names of all the unprivileged TAS commands and a brief description of each command. The privileged commands help file (pathname `"/usr/tsy/help/priv"`) contains similar information about the privileged commands. This file and the privileged command help files have the TASMMASTER as owner and an access mode of read-by-owner only. This prevents their being viewed by any TAS user except the TASMMASTER. The other help files have an access mode of read-write by all. The files other than the general help files contain a user guide description of the command whose name they bear (pathnames `"/usr/tsy/help/COMMAND"`). (See figure 3.)

The directory `"/usr/tsy/host"` contains files and directories which pertain to the statuses of the COINS II applications and to the statuses and data of the batch jobs sent to the various applications. The GHOST file (pathname `"/usr/tsy/host/ghost"`) contains the status for each host on the network, number of pending queries, and submission/response date-time-group (DTG) of the last received query response for each

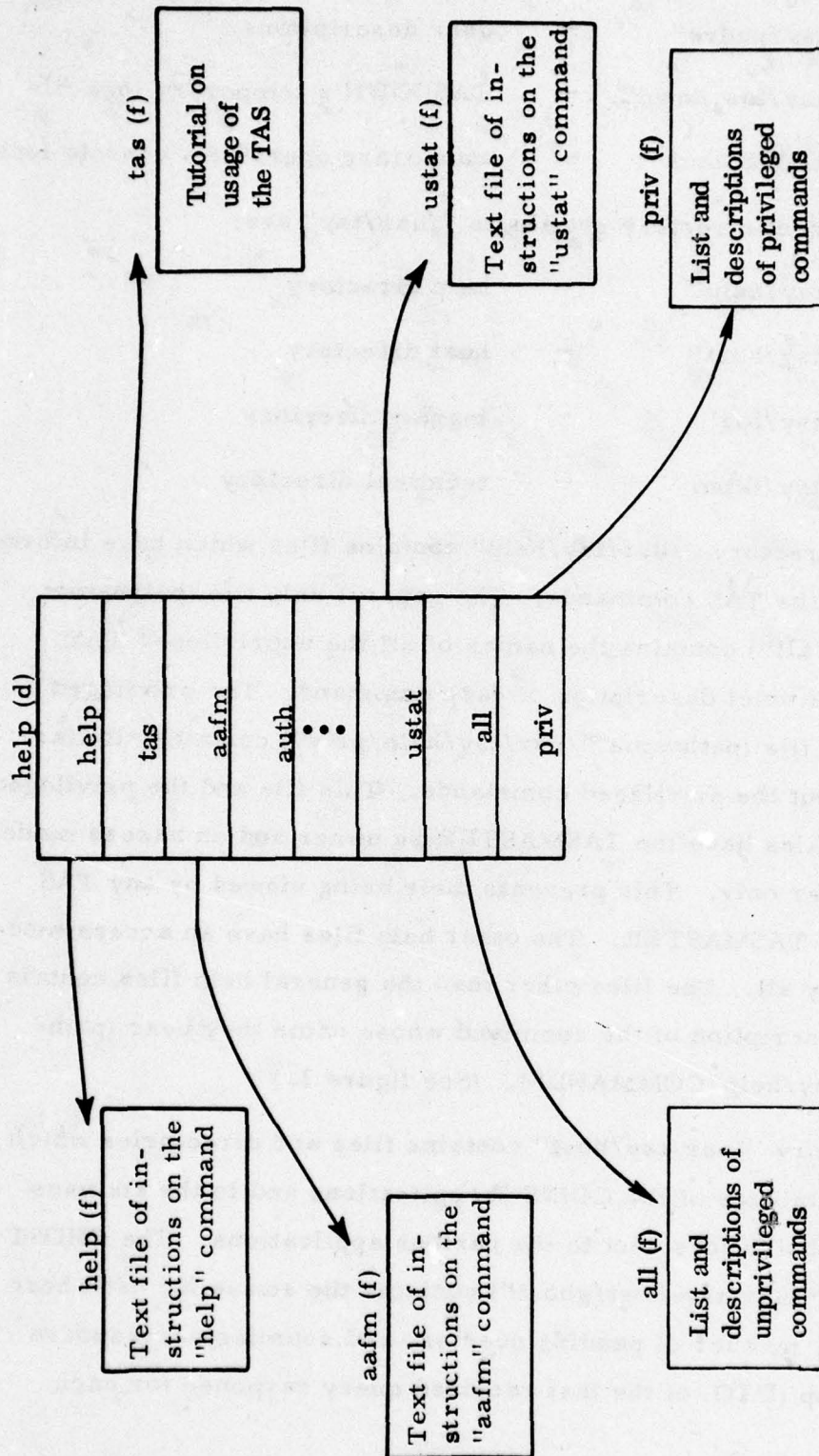


Figure 3. TAS Help Directory Entries

batch host on the network. The GPIDS file (pathname `"/usr/tsy/host/gpids"`) is a sequential file containing the command process IDs of those TAS daemons which must be externally killed by TASDOWN. The third file, GIMSG (pathname `"/usr/tsy/host/gimsg"`), contains information concerning each message sent across the COINS II network which has not yet received a RCPT. Other permanent file entries are the BQRD batch job descriptions, GJOBS, and batch query response processor pathnames, GRESP. The pathnames for these files are `"/usr/tsy/host/stat01, stat02, stat04"` and `"/usr/tsy/host/resp01, resp02, resp04"` where RYETIP is assigned the application ID of 01; DIAOLS, 02; and ISSPIC, 04. (Refer to global data description of GAPNM.) While a connection is open to a batch application, the process ID of the write daemon for that application is stored in the temporary file GIPID (pathnames `"/usr/tsy/host/pidd01, pidd02, pidd04"`). INI utilizes GIPID existence to determine whether a connection is open.

Twenty subdirectories exist in the `"/usr/tsy/host"` directory and contain the data files for batch queries and SRVC requests. The BQRD job directories (`"/usr/tsy/host/job0, job1, job2, . . . , job9"`) contain entries for the data files used by the BQRD process for batch jobs. (The GJOBS files for batch applications reference all of these data files.) The entries in the directory `"/usr/tsy/host/jobx"` are those files with JOBIDs which end with the same digit as the directory (`"/usr/tsy/host/jobx/jobid"`). The second group of subdirectories are the INI job directories. Like the BQRD job directories there are ten directories (`"/usr/tsy/host/ijb0, ijb1, ijb2, . . . , ijb9"`) and within each directory are found entries for files with JOBIDs which end with the same digit as the directory. The files are data files for INI communications with the network. There exists a one-to-one correspondence with these files and item entries in the INI batch message descriptions, GIMSG. (See figure 4.)

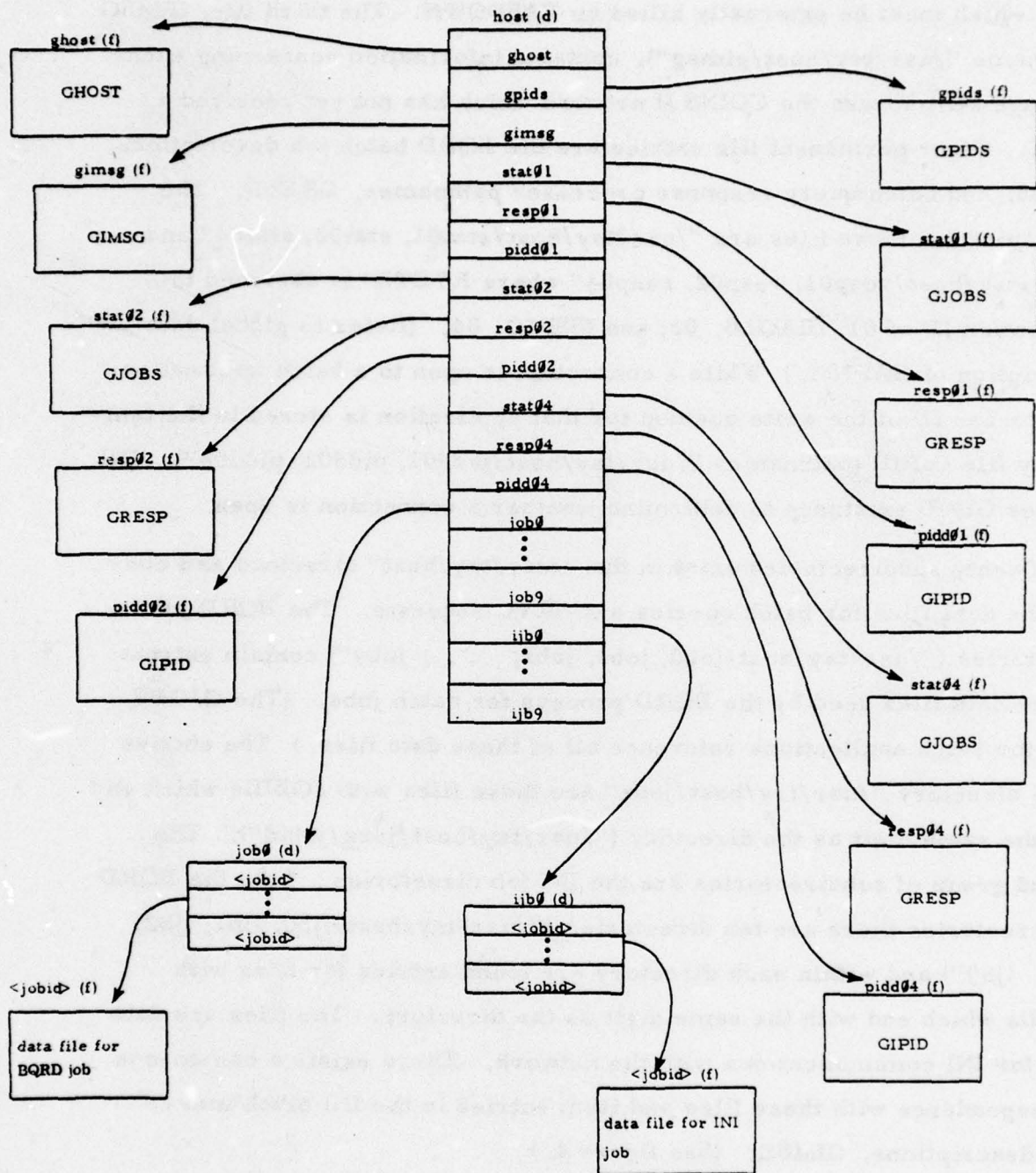


Figure 4. TAS Host Directory Entries

The directory `"/usr/tsy/log"` contains those files and directories required for the TAS logging processes, TLP. The following list describes the eight logging file entries:

1. `"/usr/tsy/log/gljobs"` — COINI record descriptions
2. `"/usr/tsy/log/glbuf"` — COINI buffer for the record logging tape
3. `"/usr/tsy/log/gtcts"` — network message traffic counts
4. `"/usr/tsy/log/tltal"` — TAS activity log, a chronological record of all significant TAS events
5. `"/usr/tsy/log/tlog"` — messages built by TLP to be output by the SPOOL process to the operator's console
6. `"/usr/tsy/log/tlogtemp"` — temporary buffer used by the SPOOL process to output TLOG
7. `"/usr/tsy/log/ltpid"` — logging tape daemon process ID
8. `"/usr/tsy/log/ltsize"` — number of current logging tape records

Within the logging directory are entries for ten subdirectories (pathnames `"/usr/tsy/log/job0, job1, job2, . . . , job9"`) which, in turn, contain entries (pathnames `"/usr/tsy/log/jobx/jobid"`) which point to COINI record files. Each file is referenced in file GLJOBS and is a fixed length record in which is accumulated the status and history of each batch interrogation or interactive session as identified by its TAS JOBID. (See figure 5.)

The last directory in `"/usr/tsy"` is the terminal directory `"/usr/tsy/term"`. Its only file entry points to the terminal descriptions file GTERM (pathname `"/usr/tsy/term/gterm"`) which has an item description for each terminal defined under TAS. Also for each terminal defined for TAS, there

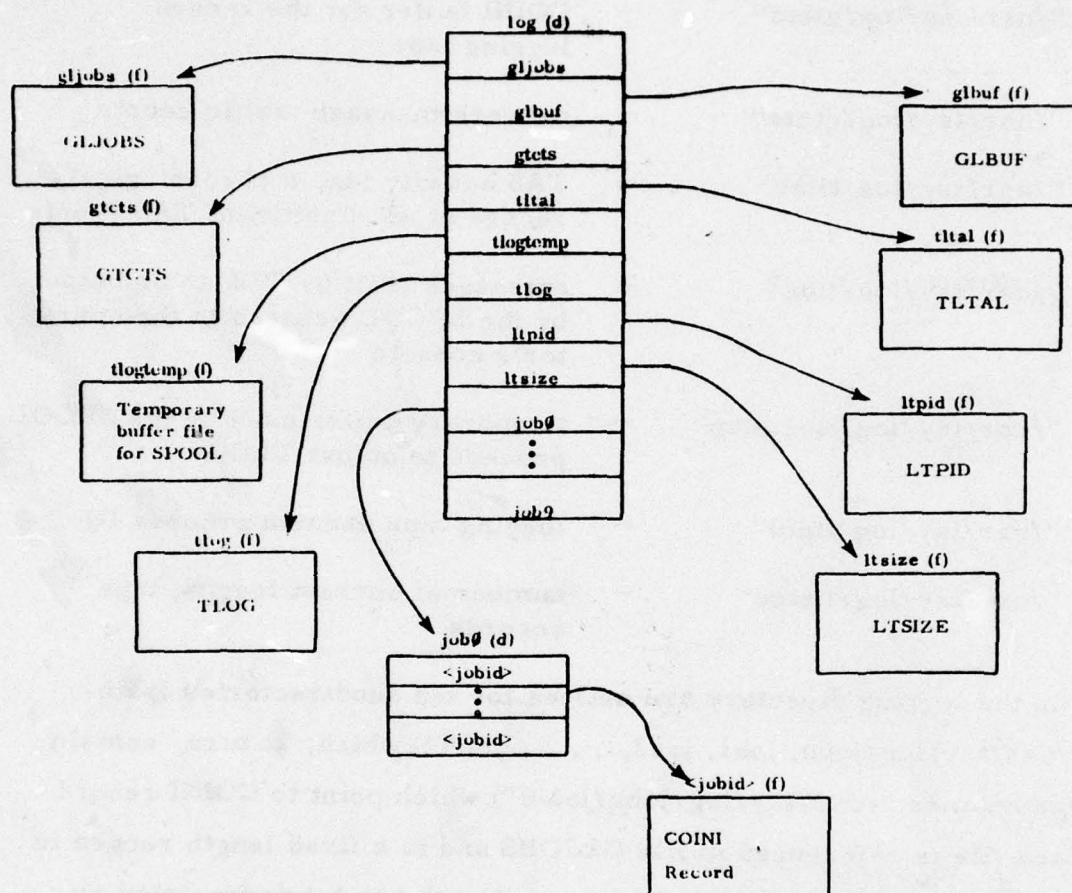


Figure 5. TAS Logging Directory Entries

exists a directory `"/usr/tsy/term/ttyn"` in which are contained those files pertaining to terminal authorization. Two permanent file entries exist: `"/usr/tsy/term/ttyn/gtaut"` which points to the terminal authorization file GTAUT, and `"/usr/tsy/term/ttyn/gappl"`, which points to the applications authorization file, GAPPL. This latter file has one item for each application which may be accessed from this terminal. Depending on the terminal's application access privileges, "ttyn" may contain one to three other entries (pathnames `"/usr/tsy/term/ttyn/fil01, fil02, fil04"`) which are the file authorization files (GFILS) for their respective batch applications. These files contain one item for each file accessible from this terminal for the specified application. The files are of variable length with the number of items in each specified in the file GAPPL. (See figure 2.)

The two UNIX directories `"/u1"` and `"/u2"` share the user directories which are subdivided, `"/u1/usr0, usr2, usr4, usr6, usr8"`, and `"/u2/usr1, usr3, usr5, usr7, usr9"`. The even numbered user directories are rooted to `"/u1"` and the odd numbered user directories to `"/u2"`. Each `"/ua/usrb"` directory accommodates those individual TAS user directories with numeric user IDs which end in the same digit, e.g., `"/ua/usrb/sni00b"`, `"/ua/usrb/sni01b"`, . . . , `"/ua/usrb/sni99b"`. (See figures 1 and 6.)

When a user is added to the TAS environment, the user directory (for the sake of illustration, pathname `"/u1/usr0/sni000"` is used) is created along with four directory entries: the query directory, `"/u1/usr0/sni000/query"`; the response directory, `"/u1/usr0/sni000/respo"`; the message directory, `"/u1/usr0/sni000/msg"`; and a TAS data directory, `"/u1/usr0/sni000/tasd"`. The basic directory (`"/u1/usr0/sni000"`) will also contain entries pointing to user defined text files as they are created by the user. The query directory contains an entry for each batch query currently

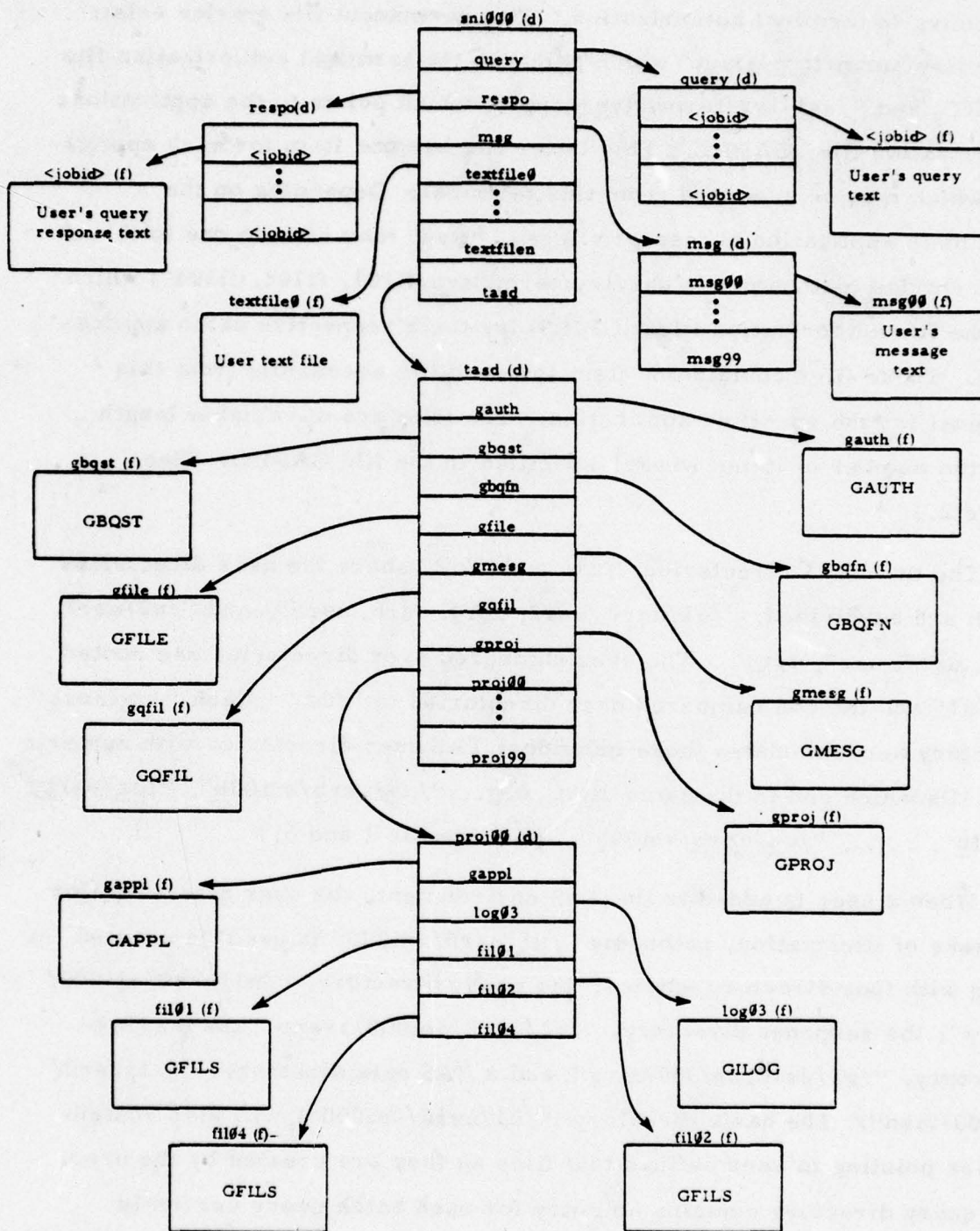


Figure 6. TAS User Directory Entries

present for the user. The actual entries in this query directory are JOBIDs assigned by the system to individual batch interrogations and point to batch interrogation text files. These files are named by their JOBIDs (pathname `"/ul/usr0/sni000/query/jobid"`). Corresponding to this directory is the response directory which contains entries for query responses. Again, it contains the actual JOBID assignments as entries where they point to the response text files as received from the batch application systems. The response text files (pathname `"/ul/usr0/sni000/respo/jobid"`) utilize the JOBID for their naming conventions. The next directory (`"/ul/usr0/sni000/msg"`) contains pointers to text messages that are currently active (i.e., have not yet been perused) for this user (pathnames `"/ul/usr0/sni000/msg/msg00, msg01, . . . , msg99"`). The final directory entry is the user TAS data directory `"/ul/usr0/sni000/tasd"`. This directory contains pointers to important TAS information concerning the user, in particular, the user's authorization data.

The directory `"/ul/usr0/sni000/tasd"` contains pointers to files and directories pertaining to user access authorization and TAS maintained user status information. Below are listed the pathnames and descriptions of those files in the "tasd" directory.

1. `"/ul/usr0/sni000/tasd/gauth"` - the user authorization file, GAUTH, contains administrative reference data for the user, including the number of projects defined in GPROJ.
2. `"/ul/usr0/sni000/tasd/gbqst"` - user batch query status file, GBQST, contains one item for each batch query currently present for the user.
3. `"/ul/usr0/sni000/tasd/gbqfn"` - user batch query filenames, GBQFN, contains one item for each batch host file referenced by the user in a batch query. GBQFN is a permanent file used in conjunction with GBQST.

4. `"/ul/usr0/sni000/tasd/gfile"` - user text file descriptions, GFILE, contains one item for each user text file contained in directory `"/ul/usr0/sni000"` and varies in size as the user creates and purges his text files.
5. `"/ul/usr0/sni000/tasd/gmesg"` - user message descriptions, GMESG, contains one item for each message held in directory `"/ul/usr0/sni000/msg"`.
6. `"/ul/usr0/sni000/tasd/gqfil"` - batch query filenames for authorization check, GQFIL, contains one item for each batch host file referenced by the user in a specific batch query. The access authorization process (AAP) validates whether the user and terminal are privileged to access the given batch host files. The temporary GQFIL file is used to update GBQFN and is only present while a user is using a Batch Query Processor (BQP).
7. `"/ul/usr0/sni000/tasd/gproj"` - project descriptions, GPROJ, contains one item for each project which is assigned to a user. For each project listed in GPROJ, there is a subdirectory entry (pathnames `"/ul/usr0/sni000/tasd/proj00, proj01, . . . , proj99"`).

Each project described in GPROJ has a project directory, `"/ul/usr0/sni000/tasd/projxy"`, which in turn has application and file access authorization data. Each user has at least one project which is the default project, with the pathname `"/ul/usr0/sni000/tasd/proj00"`. The only permanent entry in each project directory (pathname `"/ul/usr0/sni000/tasd/projxy/gappl"`) points to GAPPL, the user's application description file for this project. The number of items in GAPPL is specified in GPROJ and there is one item for each application which the user may access while working on this project. Should the user have access to an interactive application, the project directory contains an entry which points to the logon information file, GILOG (e.g., if the host is SOLIS, the pathname is `"/ul/usr0/sni000/tasd/projxy/log03"`). This file contains the necessary data to perform distant interactive host logons for this user. Should the user have access to one or more batch applications, the project directory contains entries (pathnames `"/ul/usr0/`

sni000/tasd/projxy/fil01, fil02, fil04") which point to the file authorization data files, GFILS, for each application respectively. Each of these files contains one item for each file which the user may access for the project/application, with the number of items specified in GAPPL. Note: The default project directory need have only one entry, the pointer to GAPPL, while other project directories must have at least two entries, the pointer to GAPPL and a file pointer to either GILOG or GFILS.

TAS PROCESS FLOW AND COMMUNICATION - Figure 7 illustrates the overall architecture of TAS and depicts the relationships which exist among the processes which comprise TAS.

At the top of the diagram is a dashed line representing the user interface, above which are listed all TAS command names. These commands are interpreted by the TAS command interpreter (TASHELL) and cause one or more processes to be executed. These processes range from the command processes themselves, shown just below the TASHELL, to the TAS system processes (IQI, BQRD, INI, AAP, TLP, etc.) which are shown toward the bottom of the diagram. At the lowest level is the Network Control Program (NCP) which interfaces TAS to the network.

The relationships between the processes are depicted by lines which have the following meanings:

- A process which points to another process via a solid line creates that process using the "fork-execute" or "execute" mechanism.
- A process which points to another process via a dashed line, writes/reads data via pipes to/from that process in the direction indicated.

(Note: processes communicating with the NCP are shown using dashed lines; these however indicate system read/write calls rather than communication over a pipe.)
- Processes connected by a dotted line exchange signals in the direction indicated.

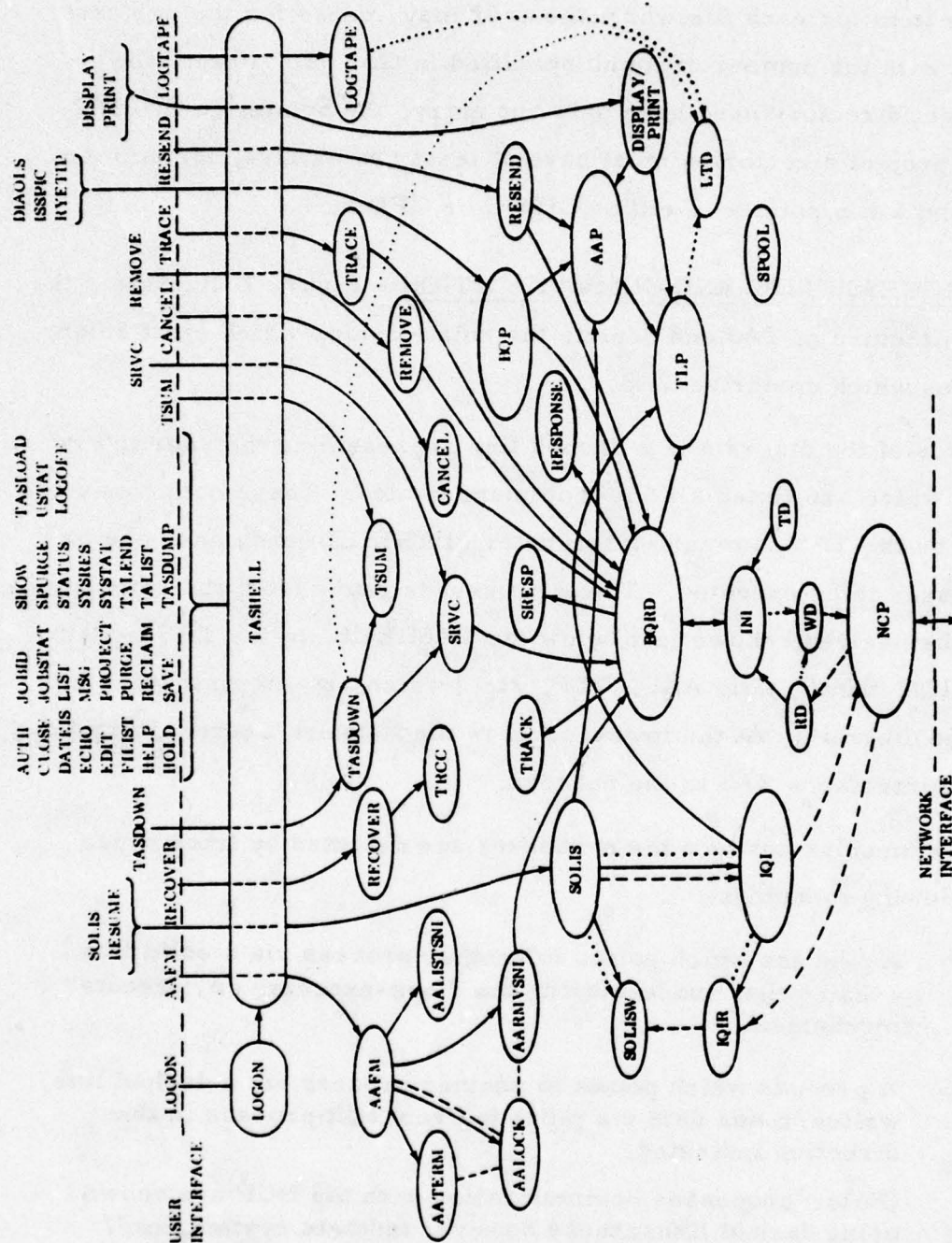


Figure 7. TAS Process Flow and Communication

TAS PROCESS DESCRIPTIONS

This section provides a description of the processes comprising the TAS. The processes, most of which are in alphabetical order, are as follows:

- | | | | |
|----|---------------|---|---|
| a. | AAFM | - | Access Authorization File Maintenance Process. |
| b. | AALOCK | - | AAFM Lock Daemon. |
| c. | AALISTSNI | - | AAFM List SNI Process. |
| d. | AARMVSNI | - | AAFM Remove SNI Process. |
| e. | AATERM | - | Terminal Access Authorization File Maintenance Process. |
| f. | AAP | - | Access Authorization Process. |
| g. | AUTH | - | Display Access Authorization Data. |
| h. | BQP | - | Batch Query Processors (RYETIP, DIAOLS, and ISSPIC). |
| i. | BQRD | - | Batch Query and Response Dispatcher Process. |
| j. | CANCEL | - | Cancel a Pending Batch Query. |
| k. | CLOSE | - | List User's Jobs for Purging on Logoff. |
| l. | DATEIS | - | Display and/or Set Date. |
| m. | DISPLAY/PRINT | - | Display/Print Batch Query Response. |
| n. | ECHO | - | Test Terminal Operability. |
| o. | EDIT | - | TAS Text Editor. |
| p. | FILIST | - | List Users' Files and Batch Queries. |

q.	HELP	-	Display HELP Information.
r.	HOLD	-	Hold a Batch Query.
s.	INI	-	Intelligence Network Interface Process.
t.	IQI	-	Interactive Query Interface Process.
u.	IQIR	-	Interactive Query Interface Read Process.
v.	JOBID	-	Display/Set TAS Jobid.
w.	JOBSTAT	-	Display Status of User's Batch Queries.
x.	LIST	-	List User's Text Files.
y.	LOGOFF	-	Log Off of TAS.
z.	LOGON	-	Log Onto TAS.
aa.	LOGTAPE	-	COINI Logging Tape Interface Process.
bb.	LTD	-	Logging Tape Daemon.
cc.	MSG	-	Send or Receive Messages.
dd.	PROJECT	-	Set User's Project.
ee.	PURGE	-	Purge a Batch Query or Text File.
ff.	RD	-	Network Read Daemon.
gg.	RECLAIM	-	File Reclamation Process.
hh.	RECOVER	-	TAS Recovery Process.
ii.	REMOVE	-	Remove a Batch Query or Text File.
jj.	RESEND	-	Resend a Batch Query.
kk.	RESPONSE	-	Batch Query Response Processor.
ll.	SAVE	-	Save a Batch Query in a Text File.
mm.	SHOW	-	Display a Text File or Batch Query.

nn.	SOLIS	-	SOLIS Interactive System Processor.
oo.	SOLISW	-	SOLIS Interactive System Write Process.
pp.	SPOOL	-	TAS Spooling Daemon.
qq.	SPURGE	-	Purge Pending Batch Jobs/COINI Entries.
rr.	SRESP	-	SRVC Response Processor.
ss.	SRVC	-	Batch SRVC Process.
tt.	STATUS	-	Display Host Statuses.
uu.	SYSRES	-	System Resource Daemon.
vv.	SYSTAT	-	Display Pending Batch Jobs.
ww.	TALEND	-	Terminate TAS Activity Log.
xx.	TALIST	-	List TAS Activity Log.
yy.	TASDOWN	-	TAS Shutdown Process.
zz.	TASDUMP	-	Dump TAS Environment.
aaa.	TASLOAD	-	Reload TAS Environment.
bbb.	TD	-	Batch Job Timeout Daemon.
ccc.	TINIT	-	TAS Initialization Process.
ddd.	TLP	-	TAS Logging Processes.
eee.	TRACE	-	Trace a Batch Query.
fff.	TRACK	-	Trace Response Processor.
ggg.	TRCC	-	TAS User Recovery (see RECOVER).
hhh.	TRCC2	-	TAS System Recovery (see RECOVER).
iii.	TSUM	-	Traffic Summary Report Daemon.
jjj.	USTAT	-	Display User Statuses.
kkk.	WD	-	Network Write Daemon.

Each process description is organized as follows. A short general description is followed by a list of the TAS global data used. Then, if applicable, a list of pertinent local data contained in the process is provided. A general data flow narrates the overall paths the process must take to accomplish its tasks. After the general flow, salient functions contained in the process are highlighted. These functions, which were called out by name in the general flow description, are described in more detail. Following the process description are flow charts which illustrate the general program paths of the process. Flow charts of the salient functions complete each process description.

AAFM (Access Authorization File Maintenance Process)

The AAFM process initializes and maintains the terminals' and users' access authorization (AA) files. The contents of the AA files determine the access capabilities of the terminals assigned to TAS and the users of the TAS environment. AAFM is called by means of the AAFM command and is a subsystem of TAS. Once the AAFM process is invoked, the user may create, update, or delete those AA files over which he has supervision. As a user types in an AAFM statement, AAFM parses this input, builds a parse tree, validates it semantically, and executes the request. If an error is detected, an informative error message is output, and generally the request is not executed. The AAFM process is terminated by the quit statement and the user is returned to the TAS Shell program.

GLOBAL DATA USAGE – AAFM uses the following global data:

- GAPPL – application authorization data**
- GAUTH – user authorization data**
- GFILS – file authorization data**
- GILOG – interactive application logon information**
- GPROJ – project descriptions**
- GUDES – user descriptions**
- GAMAP – access mode bit maps**

LOCAL DATA USAGE - The first two local data definitions refer to arrays concerned with the parsing and storing of AAFM statements. The next five mnemonics refer to frequently used indices. The last defined local variable is a flag used by AAFM.

- AMTREE - parse tree. AAFM uses the parse tree for AAFM statement processing. In this integer array predictably placed nodes correspond to pertinent elements in a statement. AMTREE is also used as a buffer when manipulating data in and out of the password file.
- AMTDAT - parse tree data. The AMTDAT character array holds data values for a specific statement. AMTDAT is indexed by pointers stored in the parse tree (AMTREE). AMTDAT also doubles as a buffer for entering or removing data in the password file.
- AMMUSID - the user ID of the user whose files are being maintained. The maintained user's (MUS) user ID is assigned when the sni is created and stored in GUES. ASCII derivatives of the user's ID are placed in pathnames associated with the MUS's AA files.
- AMADMID - the user ID of the TAS user who is the administrative user (ADM) with supervision of the MUS's AA files. An ADM exists only for ordinary users and the sni-prefixes of the MUS and ADM match. The ASCII derivatives of AMADMID are placed in pathnames associated with the ADM's AA files.

- AMPJIND - the index of the project which is currently being maintained. AMPJIND is used for the pathname associated with the project and for an index into the GPROJ file. AMPJIND is set to zero for the default project.
- AMAPIND - the index of the application which is currently being added or changed. AMAPIND is used for the index into the GAPPL file.
- AMFLIND - the index of the file which is currently being added, changed, or deleted. AMFLIND is an index into GFILS.
- AMBRFL - the branch flag which is true whenever the statement currently stored in AMTREE and AMTDAT has to be further processed. The need for the flag is due to AAFM's hierarchical, structured processing of projects, applications, and files.

GENERAL PROCESS FLOW - AAFM, a subsystem of TAS, is activated whenever a user has input the AAFM command. The user or maintenance person (MP) is validated to be either the TASTMASTER or an administrative user, and if not, an error is output and an exit is made back to the TAS Shell. Otherwise AAFM retrieves and stores (both in ASCII and numeric form) two sets of pipe (read/write) file descriptors for the purpose of establishing a mechanism for AAFM processes to communicate with the lock daemon AALOCK. After initiating the AALOCK daemon (which is briefly discussed below), AAFM sets its own current process user ID to the super-user.

Due to AAFM's language complexity, security access validation completeness, and global data requirements, the AAFM statements are processed by a combination of four processes and a daemon. The "main" function of the AAFM process is to act as the front-end dispatcher of the AAFM workload in the following manner:

- AAFM - Besides working as a front-end processor, AAFM creates and maintains user's AA files with the additional responsibility of overseeing the administrative/ordinary users relationship. The QUIT and EXECUTE commands are also processed by AAFM.
- AATERM - AATERM creates and deletes the terminal AA files, and maintains and lists the contents of the terminal AA files. The AATERM process is also called when an EXECUTE TERM command is processed.
- AALIST - The AALIST process lists the contents of users' AA files.
- AARMVSNi - The AARMVSNi process deletes the files and directories which are associated with a given user.
- AALOCK - Although the AALOCK process does not process any AAFM statements, it is the mechanism by which the AAFM processes lock global data files such as GUDes, GTERM, and so on. Since directories are created by the AAFM processes, their effective and real user ID is the UNIX super-user. If the UNIX super-user were to attempt to lock a file via the global function GFLOCK, the lock would be accomplished whether the file was already locked or not

(i.e. writing on a read-only file is legal for the UNIX super-user); this completely nullifies the purpose of GFLOCK. The solution to this dilemma is to have AAFM initiate the AALOCK daemon which sets its UNIX process owner to the global TAS user identifier, TUID. Having done this, the daemon sleeps until awakened by the interrupt signal QUIT or until terminated by the KILL signal. To lock a file, the AAFM processes awaken AALOCK and instruct the daemon via a pipe to lock a specified TAS global file.

The program descriptions of AATERM, AALIST, AARMVSNI, and AALOCK directly follow the AAFM description and flowcharts. The remainder of this description concentrates on the specific program logic of the TAS process AAFM.

AAFM calls the LXINIT function to parse the next incoming user statement and to build the parse tree (AMTREE) and the parse tree data (AMTDAT). Upon return to the main function, a switch is made on the statement type. Eleven statement types are acceptable and further processed; all others cause an error message to be printed out and a new statement to be processed.

Two basic validation tests must be made before AAFM brings in a copy of AATERM to process terminal statements. First, the statement cannot appear within an sni-type execute file and second, the MP must be the TAS-MASTER. If either condition is not met, an error message is output. There are five terminal statement types: CREATE TERM, CHANGE TERM, LIST TERM, DELETE TERM and EXECUTE TERM. Upon return of process control to AAFM from AATERM, a check is made on the return status. If the status is QUIT (meaning that a QUIT statement was processed by AATERM), AMTREE and AMTDAT are unlinked and AAFM exits via AMERROR.

The LIST statement results in the calling of the AALIST process.

The QUIT statement causes the parse tree (AMTREE) and the parse tree data (AMTDAT) to be unlinked from the MP's current directory and the AAFM process to exit via the AMERROR function.

The EXECUTE statement causes the UNIX external variable FIN to be set to the MP's edit file which is specified in the EXECUTE statement. An EXECUTE statement cannot appear within the contents of an execute file; i. e. , nested execute files are not permitted.

The create-sni statement is processed by the AAFM function AMCRS; the change-sni statement by AMCHS; and the delete-sni statement by AMCHS which in turn calls the AARMVSNi process.

At the completion of one of the aforementioned processes or functions, a check is made for an end-of-file condition. If it exists, FIN is set back to the MP's terminal. AAFM again calls the LXINIT function to parse the next AAFM statement and the described procedure is repeated. (See figure 8 for data flow.)

Before describing other functions, three terms should be defined:

- a. The user whose AA files are being created, updated, or deleted is referenced as the maintained user (MUS).
- b. The administrative user of the MUS is called the administrative person (ADM).
- c. The person currently using AAFM to maintain AA files is called the maintenance person (MP).

MAJOR FUNCTION DESCRIPTIONS

AMERROR — The AMERROR function is called to output a message and/or to exit. The function has three input parameters: AMERRNO is a message

number index for the error descriptions file GFEDES; AMTYP, if set, signifies that the function is to tie up loose ends and exit; and AMPTR, if non-zero, is a pointer to a name which is to be inserted into the error message. AMERROR first checks to see if AMTYP is set and the AALOCK daemon exists. If both conditions are true, AMERROR sends a KILL signal to terminate the AALOCK daemon process. Next a check is made on the error type of AMERRNO. If AMERRNO is a system error (READ, WRITE, etc.), the process user ID is set to a user other than the root. (This allows the function GFERROR to lock the operator's console safely.)

If a message is to be output, the global routine GFERROR is called to do so. If variable AMTYP is set, the user is also sent an instructional message and the AAFM process terminates; otherwise, AMERROR returns control to the calling function.

AMCRS (Create Sni AA Files) — AMCRS processes the create-sni-command block as the block is defined in Appendix A - AAFMILanguage. AMCRS not only ensures that duplicate sni names are not defined but also controls the statement types allowed in a create-sni-command block. The input sni is validated to be exactly eight characters in length and to be a unique sni in GUEDES. If both conditions are met, GUEDES is locked; otherwise, an error message is output and AAFM waits for a user input.

If the create statement has the keyword AD appended to it, an administrative user is being created. The MP has to be the TASMASTER and the input sni-prefix must be a unique sni-prefix in GUEDES. If, however, the MUS is intended to be an ordinary user, the administrative user for the input sni has to already exist. Only the TASMASTER or the appropriate administrative user can create an ordinary user. Next, the sibling link chain, as

stored in GUDES in field GLINK, is established for the newly created user. The first empty slot in GUDES is located and the new sni with the status (GLOGG) of CREATE/DELETE is written into this slot. GUDES is unlocked.

Next the password file is locked, the MUS's logon data is appended to the end of the file, and the file is unlocked.

The basic user directories and files, listed below, are established -

Directories:

SNI	(pathname "/uz/usrx/sniabx")
TAS DATA	(pathname "/uz/usrx/sniabx/tasd")
RESPONSE	(pathname "/uz/usrx/sniabx/respo")
QUERY	(pathname "/uz/usrx/sniabx/query")
MESSAGE	(pathname "/uz/usrx/sniabx/msg")

Files: (all entries in the TAS DATA directory)

GAUTH
GPROJ
GFILE
GBQST
GMESG
GBQFN

Having created the MUS and the file structure associated with the MUS, AMCRS calls LXINIT which parses the next incoming statement and builds the AMTREE and AMTDAT files. AMCRS switches on the statement type and processes it as follows:

1. Add project - Call AMAPRJ to add a project.
2. Add application - Call AMAPRJ to first create the default project and then to add an application to the default project.

3. End - If the default project has not yet been created, call AMAPRJ to establish the default project. Once the default project exists, call AMSEND to terminate the create-sni-command block.
4. Add compartment - Call AMCOM to add compartments.
5. Personnel information - Call AMBS to process the following statement types: telephone, address, name, clearance, SSN, password, utype, organization, and SNI.
6. Other statements - Output an error message.

If one of the called functions (e.g., AMAPRJ) had set the reprocess-statement flag AMBRFL, AMCRS returns control to the statement type switch (discussed above); otherwise, the LXINIT function is called to syntactically validate and parse the next incoming statement.

AMCHS (Change or Delete SNI AA files) - After locking GUDS, the input sni is validated to determine whether the following conditions are true.

1. The sni is an existing sni in GUDS.
2. The MP has the authority to alter the AA files for this sni.
3. The sni belongs to a user whose GLOGG status is logged off unless the MUS is the TASMASTER.
4. If the MUS is an administrative user, his subordinates are also logged off the system.

If any of these conditions is false, an appropriate error message is output, GUDS is unlocked, and AMCHS returns to the calling function.

If the statement being processed is a delete-sni statement, two additional checks are made: (1) the MUS cannot be the TASMASTER, and (2) if the MUS is an administrative user, there cannot be any subordinates defined for his agency-group. When these conditions are met, the MUS GLOGG status

is set to CREATE/DELETE and the remove sni process AARMVSNI is called to delete the user from the TAS environment.

When the change-sni statement is processed, the MUS's GAUTH and GPROJ files are read. If the MUS is an administrative user, the GLOGG statuses for all his subordinates are set to UPDATE. The GLOGG status is likewise set for the MUS. GUDS is then written and unlocked via GFUN-LOCK. AMCHS switches on the next input statement type and processes it as follows:

1. Add project - Call AMAPRJ to add a project.
2. Change/Delete project - Call AMUPRJ to update the project.
3. Add application - Call AMAPRJ to first create the default project and then to add an application to the default project.
4. End - If the default project has not yet been created, call AMAPRJ to establish the default project. Once the default project exists, call AMSEND to terminate the change-sni-command block.
5. Compartment - Call AMCOM to add or delete compartments.
6. Personnel information - Call AMBS to process the following statement types: telephone, address, name, clearance, SSN, password, user type, organization, and SNI.
7. Other statements - Output an error message.

AMCHS branches back to the statement type switch which was discussed above; otherwise, the LXINIT function is called to syntactically validate and parse the incoming statement.

The AMCHS function returns to the main function after processing the end statement or end-of-file condition.

AMSEND (Terminate SNI Block) - AMSEND writes out the GAUTH and GPROJ files and locks and reads GUDS. If the user is the

TASMASTER, the status is returned to logged on; otherwise, the GLOGG status is set to logged off. If the MUS is an administrative user, the GLOGG statuses of his subordinates are set to logged off as well. GUES is then written and unlocked and AMSEND returns.

AMBS (Process Basic Sni Personnel Data) - The sni basic personnel information type statements are processed by the AMBS function. This function is called by either the AMCRS (create sni function) or AMCHS (change sni function). Eight of the basic personnel data are stored in the GAUTH file, while the password file contains the user's sni and password.

When the AMBS function is called, a switch is made on the statement type. The statements, their validation, and their processing are listed below.

1. Name statement - The name of the individual for whom the sni AA files exist is given in the form of a character-constant which cannot exceed thirty characters. The name is stored in the character array GAUTH.GUNAM.
2. Clearance statement - The single-letter input clearance code is checked to determine whether it is a valid security clearance level. The bit map for the clearance is stored in GAUTH.GUCLR. If the MUS is an ordinary user, his ADM must have a clearance greater than or equal to the input clearance. If an administrative user has his clearance reduced (or changed with respect to SI and/or TK), function AMSUBOR checks and reduces, if necessary, the clearances of his subordinates.
3. SSN statement - The MUS's social security number is input as a character-constant of maximum length of nine characters. It is stored in a character array GAUTH.GUSSN.
4. Utype statement - The user type of the MUS is one of three types: OPER (operational), CRISIS, or STUDENT. These three mnemonics are the only accepted input formats. The information is stored as a number code in GUES.GUTYP.
5. Organization statement - The organization with which the user is associated is input as a character-constant with a maximum length of ten characters. This data is stored in GAUTH.GUORG.

6. Address statement - The address is input by means of a character-constant whose length cannot exceed one hundred characters and is stored in the character array GAUTH.GUADR.
7. Telephone statement - The telephone number is input as a character-constant with a maximum length of twenty characters. This number is stored in the character array GAUTH.GUTEL.
8. Password statement - The UNIX password file `"/etc/passwd"` is altered whenever a user's password is changed by the password statement. The password file is locked, opened, and read. (Data manipulation of the password file is done by a single-character reading, analyzing, and writing procedure. The data buffering is done between the password file and a temporary file.) At the beginning of each new line of the password file, a search for a match is made on the user's original sni and the sni within the line. If a match is not made, the file is read until the next new line character is read and another match is attempted. If an end-of-file is reached without a successful match, an error message is output and the function returns. If a match is made, the previous password is stripped from the file and the new encrypted password is inserted into the password file.
9. SNI statement - The changing of an administrative user's sni is similar to the changing of a user's password. The following conditions must be true before the user description file GUDES and the password file `"/etc/passwd"` are updated with the new sni.
 - a. The proposed sni must have exactly eight characters.
 - b. The MUS must be an administrative user.
 - c. The old and proposed sni-prefixes must be identical.
 - d. The proposed sni must be unique.

If the above conditions are true, the user's entry in the password table is located and the new sni is substituted for the old sni. The new sni is also stored in the character arrays GAUTH.GUSNI and GUDES.GSNI.

After AMBS has processed one of the above statements, it returns control to the calling function.

AMCOM (Process Compartment Names) — The function AMCOM processes the add and delete compartment statements. Each user may have a maximum of twenty compartments assigned to him. The AMCOM fully processes each valid compartment name in the list of input names, and outputs an error message for each invalid name. A compartment name has a maximum length of six characters. If the MP is adding a new compartment name, the name must differ from existing compartment names. In contrast, when the MP deletes a compartment name, the name must be an existing compartment name. The list of compartment names is stored in a two-dimensional array, which is a variable length list of six character compartment names. A count is kept of the number of currently defined compartment names.

AMAPRJ (Add Project Data) — Projects are added to the sni's AA files by the AMAPRJ function. The input project name is validated to be a unique project name in the GPROJ file. If the MUS is an ordinary user, his ADM must have access to the project. The project index, AMPJIND, is determined to be the first empty slot in GPROJ. The project directory is established and the project file GAPPL is created.

The AMAPRJ function calls the LXINIT function, which parses the next incoming statement, and switches on the statement type. The add-application statement causes the AMAAPP function to be invoked; the end project statement and end-of-file condition cause the AMEPRJ function to be called; and all other statements cause an error message to be output. On AMAAPP's return of program control to AMAPRJ, a test is made on the reprocess-statement flag AMBRFL. If set, the AMAPRJ function process flow returns to the statement type switch; otherwise, the flow returns to the LXINIT function call. AMAPRJ returns to the calling function after the processing of the end project statement. The program flow varies from that

described in the above paragraph in order to handle the default project. The AAFM language requires neither an add-project statement nor an end project statement in the definition of the default project. When the default project flag is set, the AMAPRJ function bypasses the call to the LEX/parser and processes the statement already stored in AMTREE and AMTDAT. A switch is directly made on the statement type. As indicated above, the add-application statement causes the AMAAPP function to be called. However, all other statements cause (1) the function AMEPROJ to be called to write GAPPL, (2) the default flag to be cleared, (3) the branch flag AMBRFL to be set, and (4) a return to the calling function.

Note: It is inappropriate to define named projects for the TASMASTER. The TASMASTER can have a default project as a mechanism to have interactive host logon data stored for him.

AMUPRJ (Update Project Data) -- A project is deleted or updated by the AMUPRJ function. The input name must be an existing project name in the sni AA file GPROJ.

If the statement type is "delete", the function AMRPRJ is called to remove the project. Next, AMUPRJ returns to the calling function.

If the statement type is "change", file GAPPL is read in and the function LXINIT is called. After LXINIT parses the next incoming statement and returns controls to AMUPRJ, a switch is made on the statement type. Below is described the processing:

1. End project - The AMEPRJ function is called to validate the project definition. AMEPRJ is also called in an end-of-file condition.

2. Add application - The AMAAPP function is called to add a new application to the project definition.
3. Delete/update application - The AMUAPP function is called to delete or update applications defined for that project.
4. Other statements - The occurrence of other statements causes the output of an error message.

Upon return of program control to AMUPRJ, a test is made on the reprocess-statement flag AMBRFL. If set, the AMUPROJ function process flow returns to the statement type switch; otherwise, the flow returns to the LXINIT function call.

After processing the end project statement or end-of-file condition, the AMUPRJ function returns to the calling function, AMCHS.

Variations on the above process flow for the default project condition can be deduced from the thorough discussion of a similar situation in function AMAPRJ.

AMEPRJ (Validate Project Data) - The termination of the creation or the maintenance of a project is done by an end project statement. The AMEPRJ function calls the AMEAPP function which completes the processing of the last application being defined. When AMEAPP returns, the AMEPRJ function validates either that the project is the default project or that at least one application is properly defined. If either condition is met, the GAPPL file is written out and the project entry is updated in the GPROJ file; otherwise, AMRPRJ is called to remove the project. A return is made to the calling function, AMAPRJ or AMUPRJ.

AMRPRJ (Remove Project Data) - The project entry in the GPROJ file and the files associated with the project are deleted. The AMRPRJ function returns to the function that called it, AMCHS. If the MUS is an administrative user, the function AMSUBOR is additionally called to remove the same project and associated files from the AA files of his subordinates.

AMAAPP (Add Application) – The function AMAAPP adds an application to the MUS's AA files. The application name in consideration cannot be an application already existing with respect to a project index AMPJIND. If the MUS is an ordinary user, the application must be defined for the ADM as well. The application index AMAPIND is set to the first available slot in GAPPL.

If the application is interactive, AMAAPP creates GILOG and calls AMIAPP.

If the application is batch, the AMAAPP function creates GFILS and calls LXINIT which parses the next input statement. When LXINIT returns, AMAAPP switches on the statement token type. If the statement is an add-file statement, it calls the AMAFIL function which processes the statement. Otherwise, it calls the AMEAPP function (which validates the condition of the application definition), sets AMBRFL to true, and returns to the calling function, either AMAPRJ or AMUPRJ. The AMAAPP function continually processes add-file statements until another statement type is input.

Note: It is inappropriate to define batch applications for the TASMMASTER.

AMIAPP (Interactive Application) – The logon interactive data are stored as character arrays in that application's file in the format described in GILOG in the order the list is input. After writing out GILOG and adding the application to the GAPPL file, the AMIAPP function returns to the calling function.

AMUAPP (Update Application) – The function AMUAPP is called whenever an application is being either changed or deleted. The AMUAPP function picks up the application name and determines whether it is currently defined in GAPPL for project index AMPJIND. If not, an error message is printed out and AMUAPP returns to AMUPRJ.

If the statement type is "delete", the application is deleted via the AMRAPP function.

When changing interactive applications, the function AMIAPP is called.

If the application is batch and the statement type is CHANGE APP, the LXINIT function is called to parse the next incoming statement, and upon its return, a switch is made on the statement type. The add-file statement causes the AMAFIL function to be called; likewise, a change- or delete-file statement results in the calling of the AMUFIL function. All other statements cause 1) a call to AMEAPP (which validates the existence of the application's necessary data), 2) the reprocess-statement flag AMBRFL to be set, and 3) a return to the calling function.

The AMUAPP function processes add-, change- or delete-file statements until another statement type is detected.

AMEAPP (Validate Application Data) — A batch application definition must be validated to insure the existence of necessary information. For each batch application at least one file must be defined. It is not, AMRAPP is called to remove the application. However, if all is in order, the application AA file GFILS is written out. The AMEAPP function returns to its calling function, either AMAAPP or AMUAPP.

AMAFIL (Add File) — A file is defined with respect to an application. An add-file statement contains a filename which is input in the form of a character-constant and is validated to be a unique name within an application definition. The maximum length of the name is ten characters.

The next step is to determine the input access mode bit map. If the MUS is an ordinary user, the file must be one to which his ADM has access and the access mode cannot be greater than his ADM's for the same file.

The AMAFIL function stores the filename and access mode in GFILS and returns to the calling function.

AMUFIL (Update File Data) – Changing or deleting information concerning an existing file is done in the AMUFIL function. The filename is validated to be an existing filename in GFILS for a given application.

If the statement type is "delete", the file data entry is removed from GFILS. The AMUFIL function returns to the calling function. If the MUS is an administrative user, a call to AMSUBOR to adjust his subordinate's AA files is made before the file is actually deleted from the MUS's AA files.

When the statement type is "change", the access mode bit map is determined. If the MUS is an ordinary user, his ADM must have greater than or equal to access rights for the same file; if the MUS is an administrative user, his access capabilities are diminished only after function AMSUBOR appropriately lessens his subordinate's AA files with respect to this particular file's access mode. The AMUFIL function stores the file data in GFILS and returns to the calling function.

AMSUBOR (Adjust Subordinates' AA Files) – The purpose of AMSUBOR is to adjust subordinates' AA files for a given administrative user in order to make those AA files compatible with the administrative user's AA files. The administrative user's access capabilities must be greater than or equal to the access authorization of his subordinates. The five areas requiring adjustments are discussed below.

1. Clearance – When an administrative user's clearance is reduced in level, a subordinate's clearance is likewise adjusted. This is done by the logical "anding" of the administrative user's new clearance and the subordinate's clearance. If necessary, AMSEND is called to rewrite GAUTH.

2. Delete Project - If the subordinate has the same project defined in his AA files, the project directory and associated files (e. g. , GAPPL) are unlinked and the project entry in GPROJ is cleared. AMSEND is called to rewrite GAUTH (project count has been decremented) and GPROJ.
3. Delete Application - If a subordinate has the same application defined for the same project, the application is removed. The files associated with the application are deleted, the application's entry in GAPPL is cleared and the application count is decremented. The function AMEPRJ is then called to insure that the subordinate's project is still properly defined (non-empty). AMSEND is called to rewrite GAUTH and GPROJ.
4. Delete File - If a subordinate has the same file for the identical application and project as his administrative user's "to-be-deleted" file, the file is deleted. The entry in GFILS is cleared and the file count is decremented. The functions AMEAPP, AMEPRJ, and AMSEND are called to ensure the proper validation and rewriting of the application and project files.
5. Reduce a File's Access Mode - If a subordinate has the same file for a given application and host defined for him, the access-mode bit map is adjusted to be compatible with that of his administrative user's new access mode.

Having analyzed and appropriately adjusted or bypassed the AA files of each member of the administrative user's agency-group, a return is made to the calling function.

LXINIT (Lex/Parse Initialization) - LXINIT is merely an initialization function for the lex/parse portion of this process. It initializes pertinent data including a flag which is returned to the calling function to indicate end-of-file (EOF), syntax error, or statement accepted. LXINIT calls the parsing function (YYPARSE) and returns with the flag value.

YYPARSE (Parser) - **YYPARSE** is a parser which is produced by the Yet Another Compiler-Compiler (YACC) program running under UNIX. YACC produces the parser as well as a set of tables which the parser uses to organize the tokens passed to it by the lexical analyzer (YYLEX). These tables reflect the grammar of the various AAFM statements. **YYPARSE** calls **YYLEX**, which returns a value called a token type. If the token type is invalid according to the input statement syntax rules, an error message is output and **YYPARSE** calls **YYLEX** continuously until an end-of-input token type is returned. **YYPARSE** then returns with the return flag set to reflect a syntax error. If the token type is valid according to the input rules, the action specified in the YACC run is performed. The actions that can be invoked are: no action, build a parse tree node with the specified number of legs, store the final token type and write out the parse tree and parse tree data, start a list, add an element to the current list, and store the list in the parse tree. After the action is performed, **YYPARSE** calls **YYLEX** for the next token type. When the end-of-input token type is received and its action performed, **YYPARSE** returns.

YYLEX (Lexical Analyzer) - **YYLEX** is the function which actually inputs and processes each AAFM statement. **YYLEX** reads and saves each input character until a delimiter is encountered.

The delimiters for AAFM are: comma, semi-colon, newline (carriage return), single quote mark, and space. If an EOF is encountered, the return flag is set to reflect EOF and an end-of-input token type is returned. If a delimiter is encountered by itself, either it is passed to the parser (comma, or semicolon), it is ignored, or it causes **YYLEX** to change its input mode; e. g., single quote mark indicates character-constant input and causes **YYLEX** to read characters until another single quote mark is input. When a delimiter terminates a string of characters, **YYLEX** must

determine what the characters represent. If the characters form an AAFM keyword, YYLEX returns with that keyword's token type. Otherwise, YYLEX determines the type of constant which is formed by the characters (name, integer, or character constant), stores the value of the constant in the parse tree data (AMTDAT), stores a terminal node in the parse tree (AMTREE), and returns with the constant's token type. If YYLEX encounters an error (e.g., too many characters input), it returns an invalid token type to the parser and thereby generates a syntax error message.

AMRAPP (Remove Application Data) – AMRAPP removes the specified application. The application file (GILOG or GFILS) is unlinked, the application's entry in GAPPL is deleted, and the MP is informed of the deletion. AMRAPP returns to the calling function.

AMAUTH (Authorize the MP) – AMAUTH determines whether the MP is authorized to create, alter or delete the files of the specified MUS. If the MP is the TASMASTER and the MUS is an administrative user or the TASMASTER, AMAUTH returns to the calling function. If the MP is the TASMASTER and the MUS is an ordinary user, his administrative user's GPROJ and GAUTH files are read and AMAUTH returns to the calling function. On the other hand, if the MP is an administrative user, the MUS's and the MP's sni-prefixes must match. If they do not, an error message is output and AMAUTH returns to the calling function with a negative value. Otherwise, the MP is established as the administrative user and his GPROJ and GAUTH files are read. AMAUTH returns to the calling function.

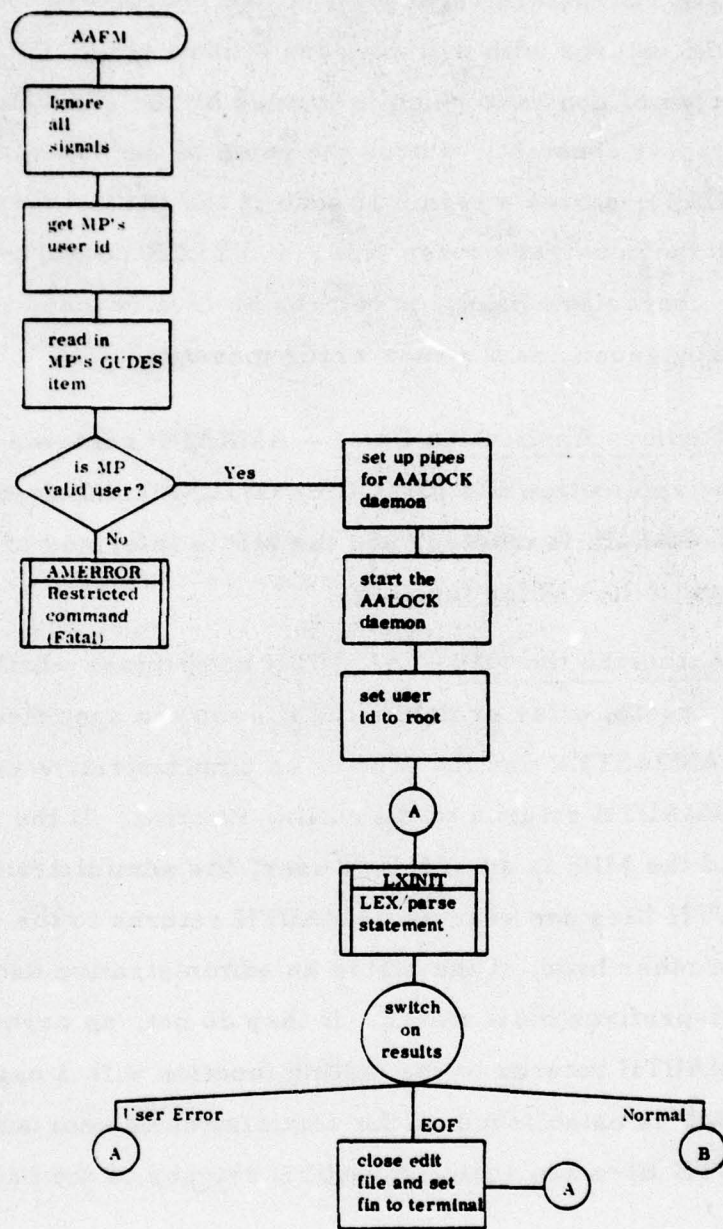


Figure 8. AAFM Process Data Flow (Sheet 1 of 32)

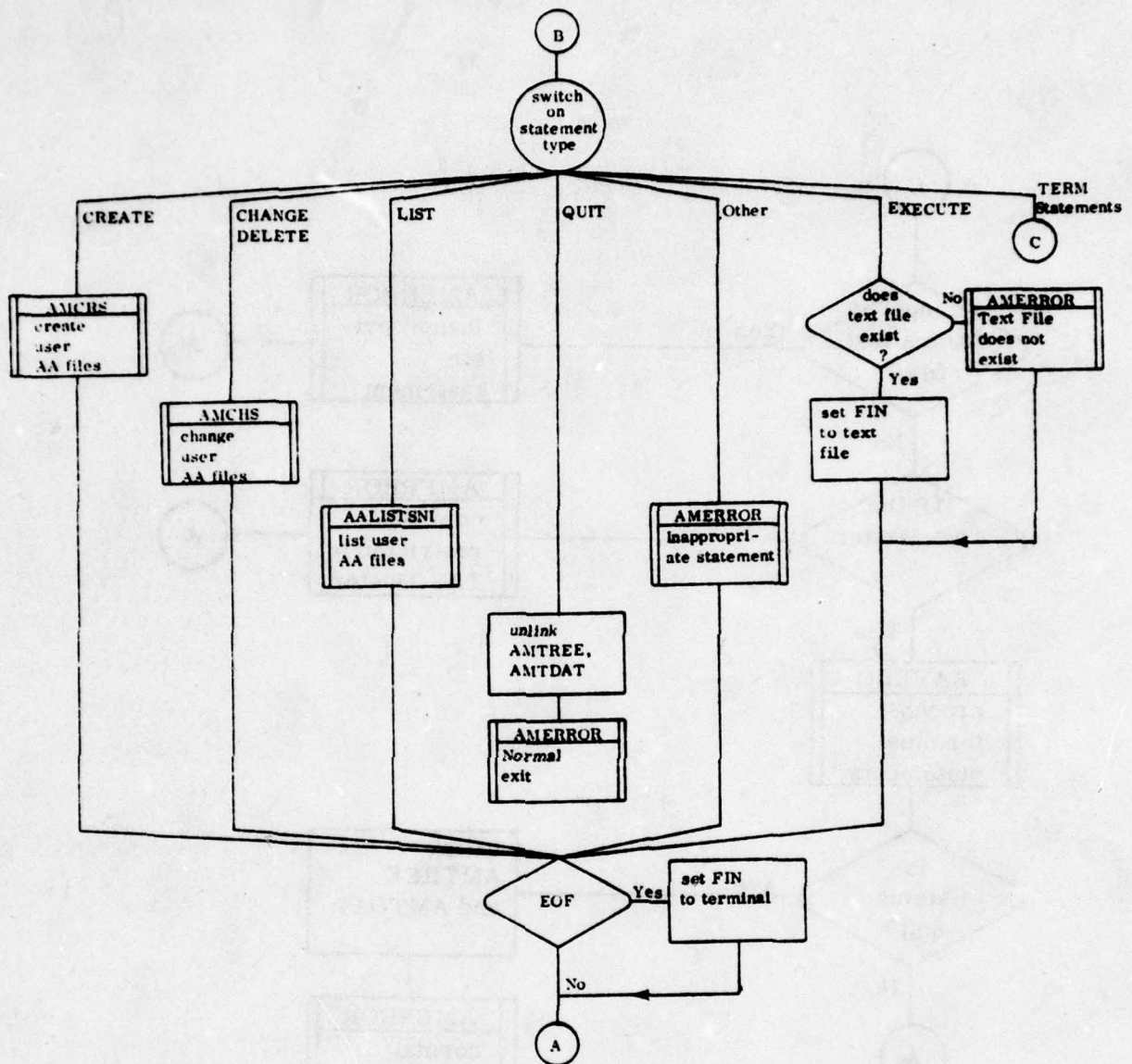


Figure 8. AAFM Process Data Flow (Sheet 2 of 32)

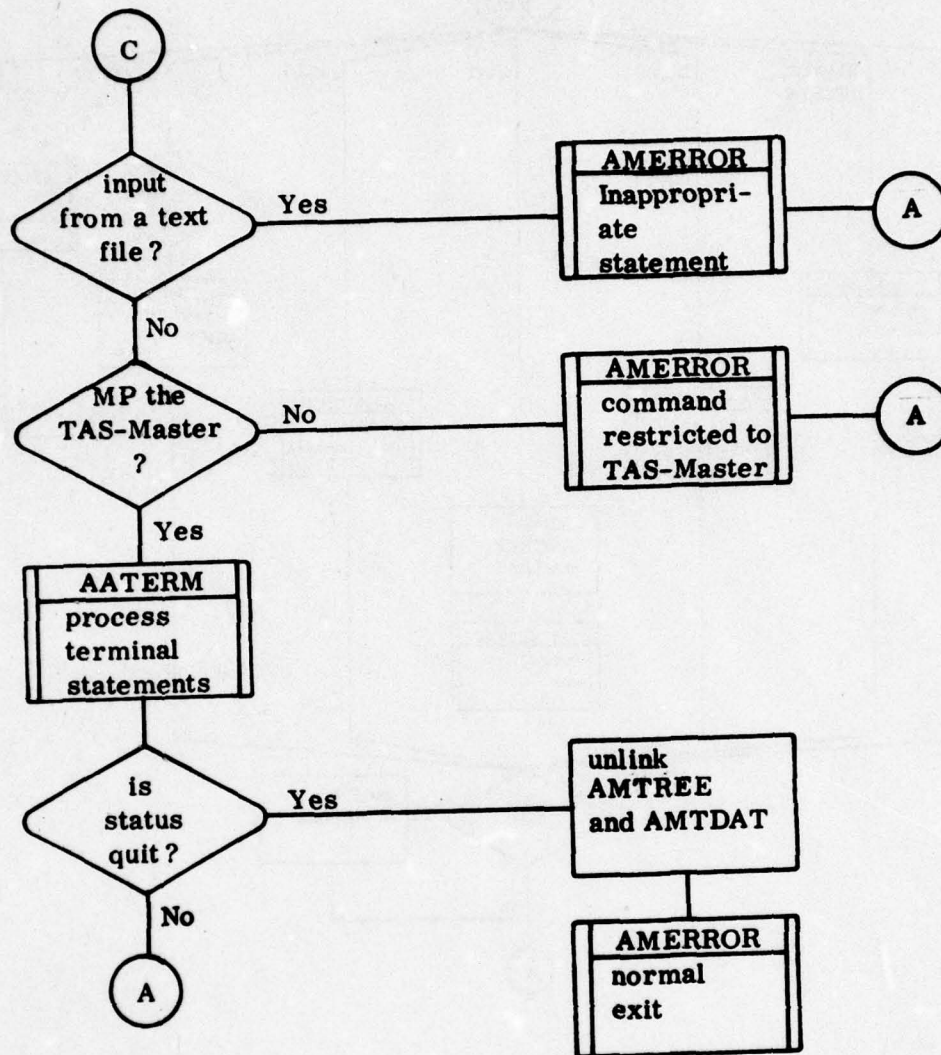


Figure 8. AAFM Process Data Flow (Sheet 3 of 32)

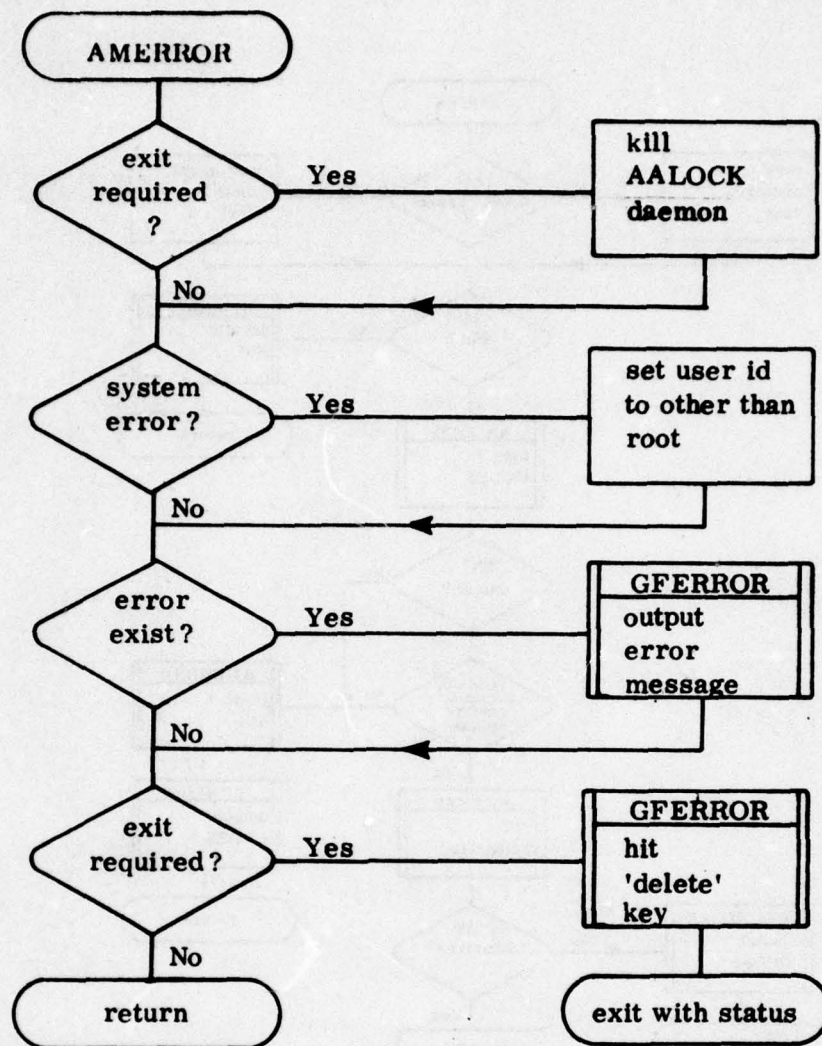


Figure 8. AAFM Process Data Flow (Sheet 4 of 32)

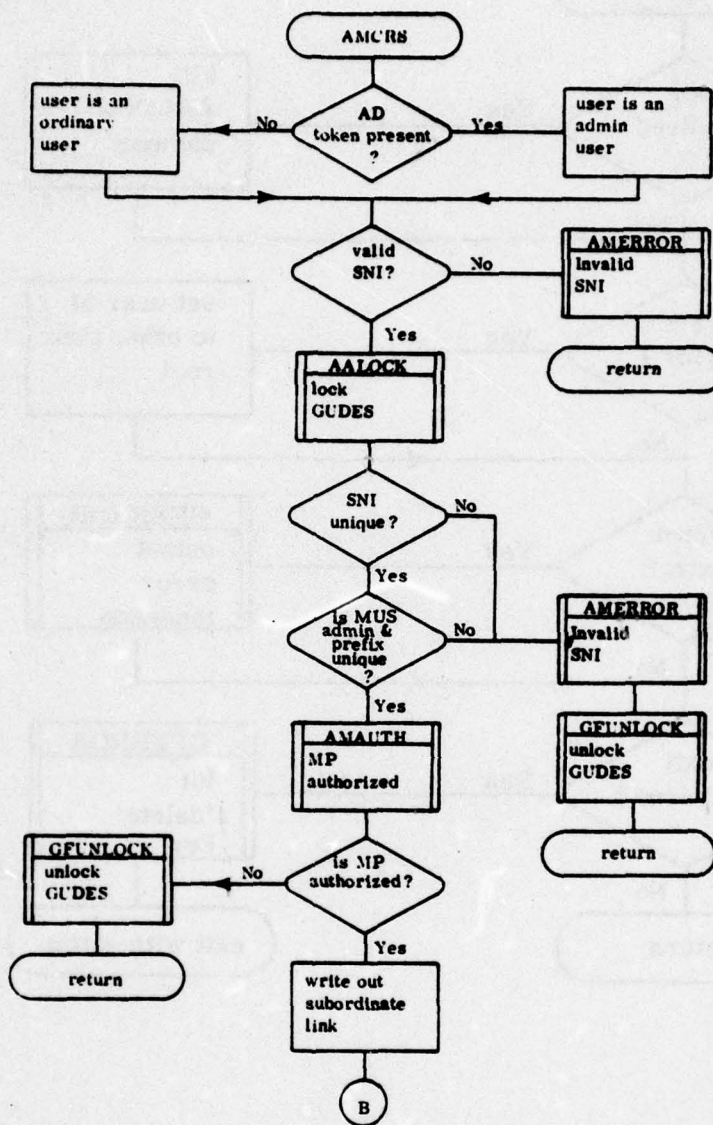


Figure 8. AAFM Process Data Flow (Sheet 5 of 32)

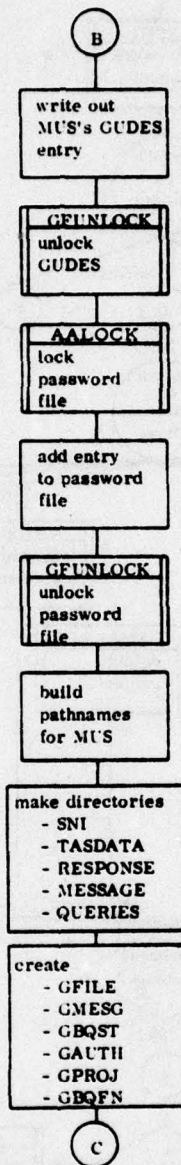


Figure 8. AAFM Process Data Flow (Sheet 6 of 32)

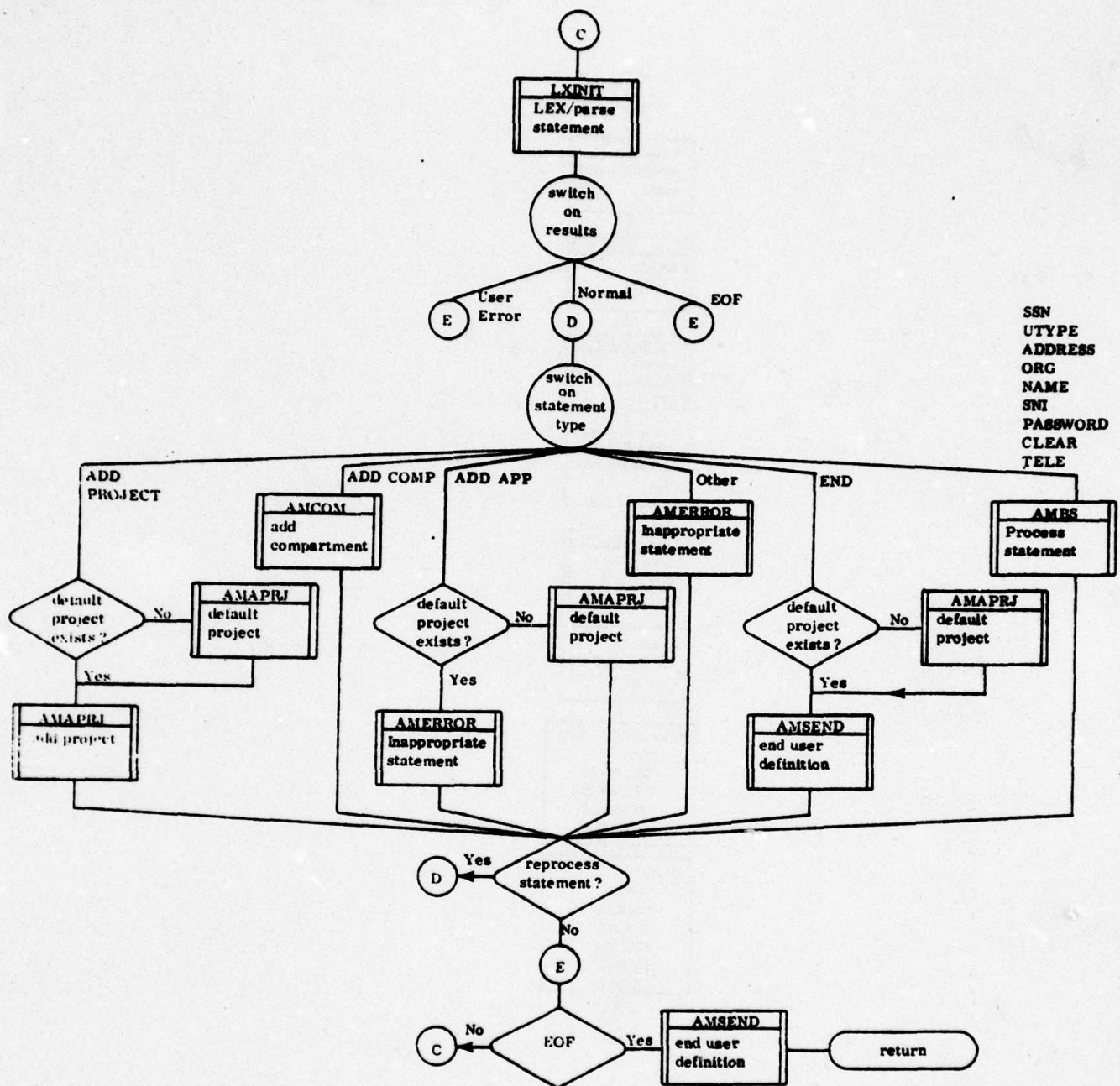


Figure 8. AAFM Process Data Flow (Sheet 7 of 32)

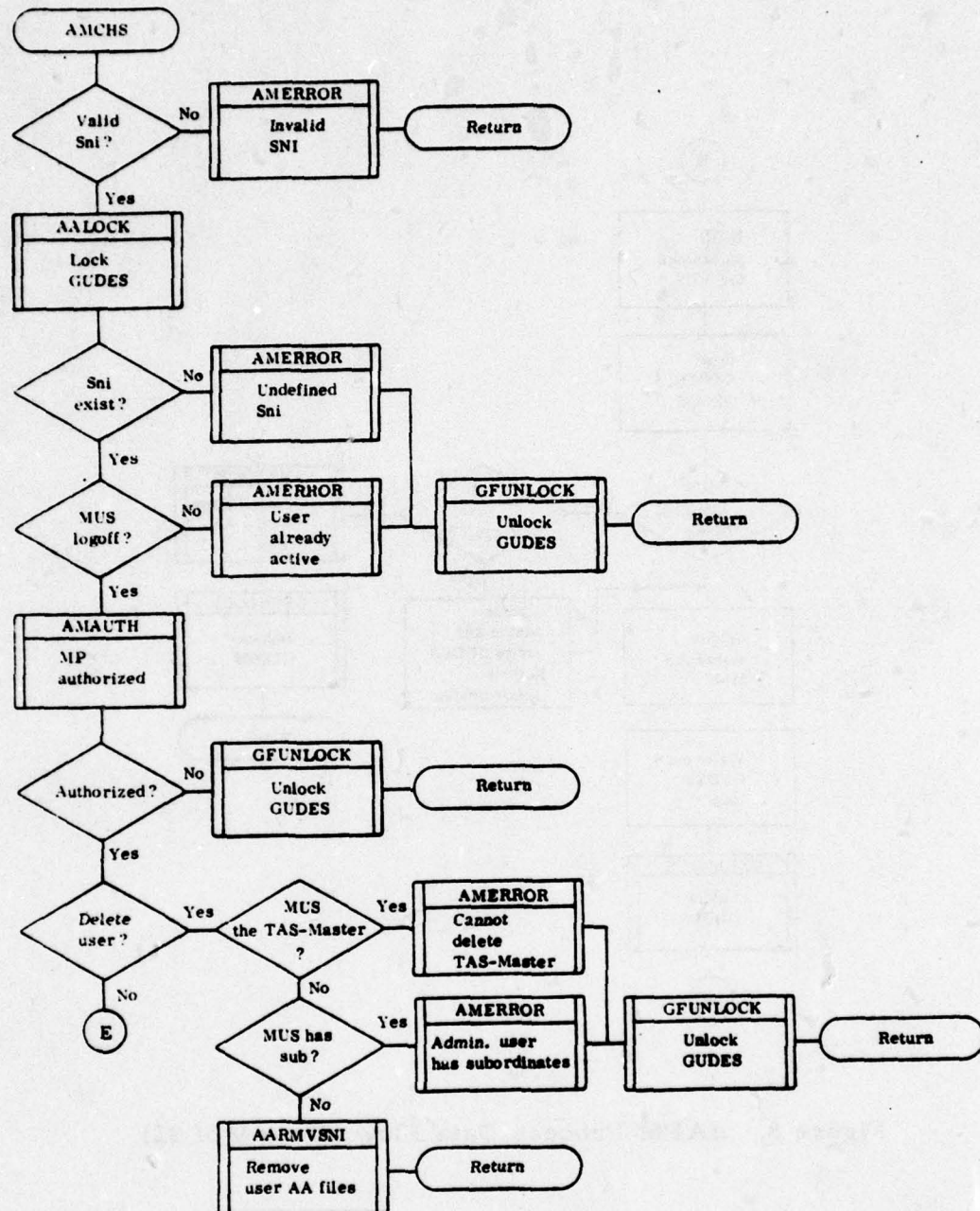


Figure 8. AAFM Process Data Flow (Sheet 8 of 32)

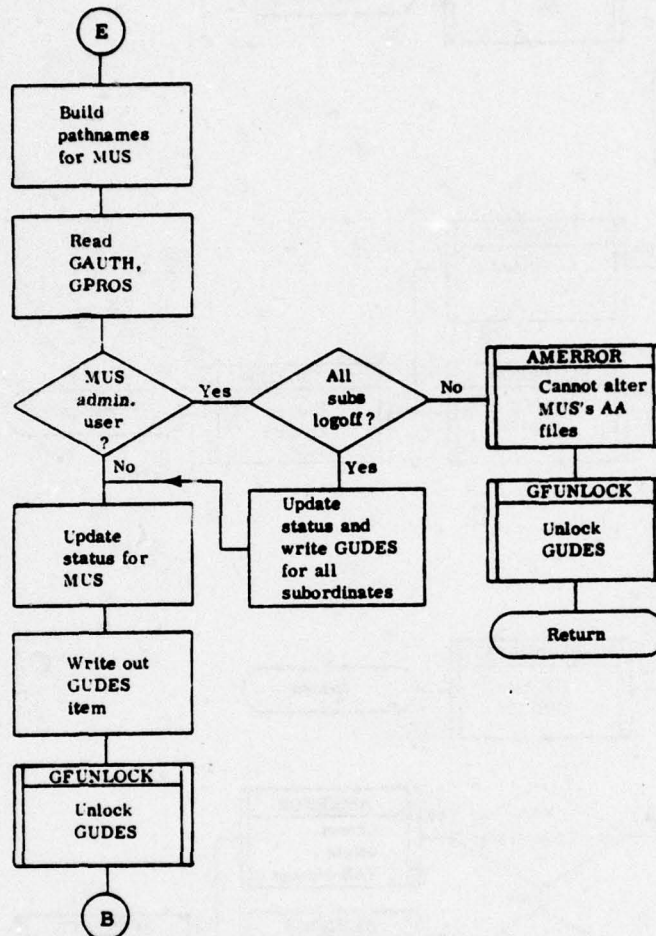


Figure 8. AAFM Process Data Flow (Sheet 9 of 32)

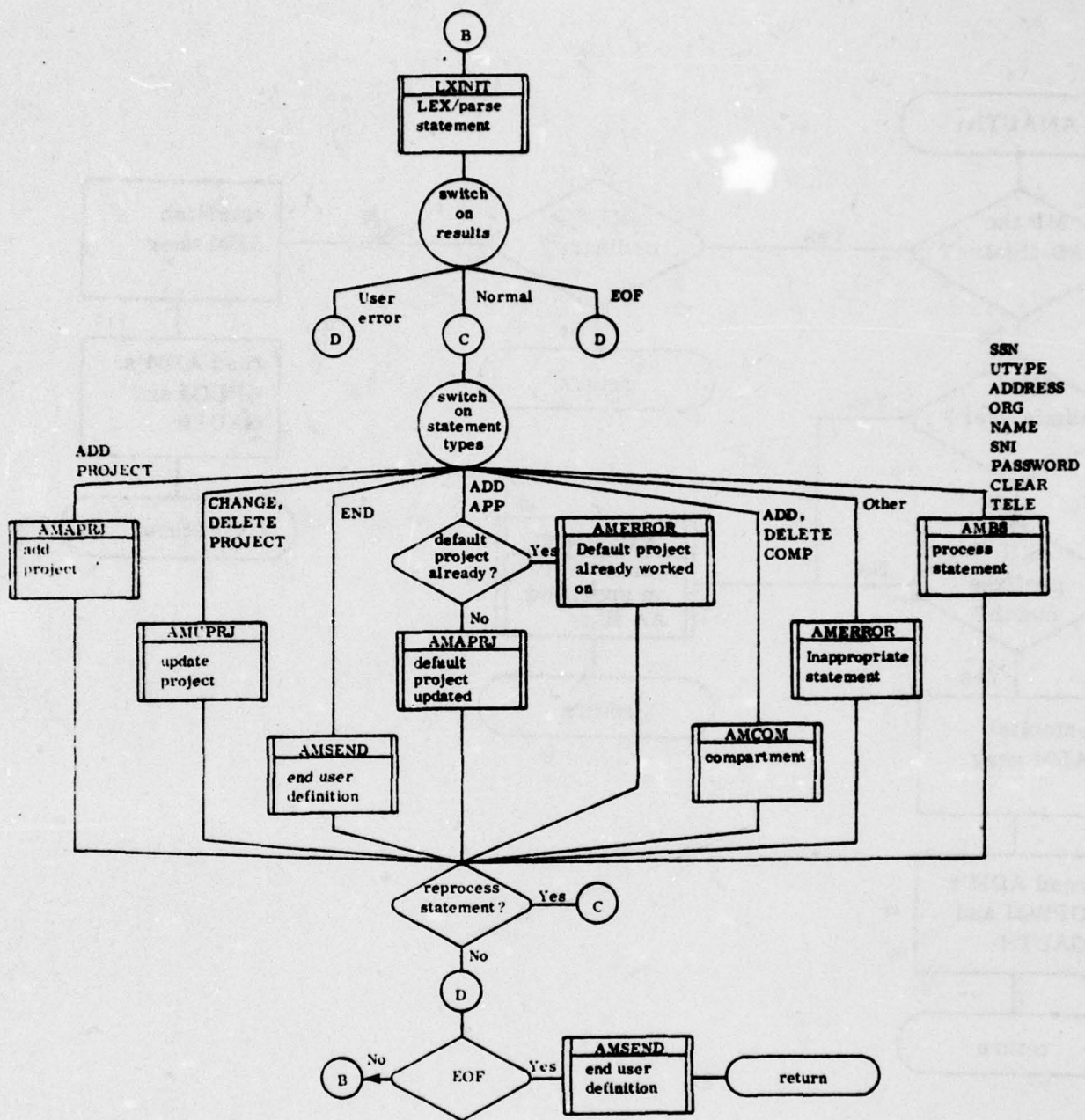


Figure 8. AAFM Process Data Flow (Sheet 10 of 32)

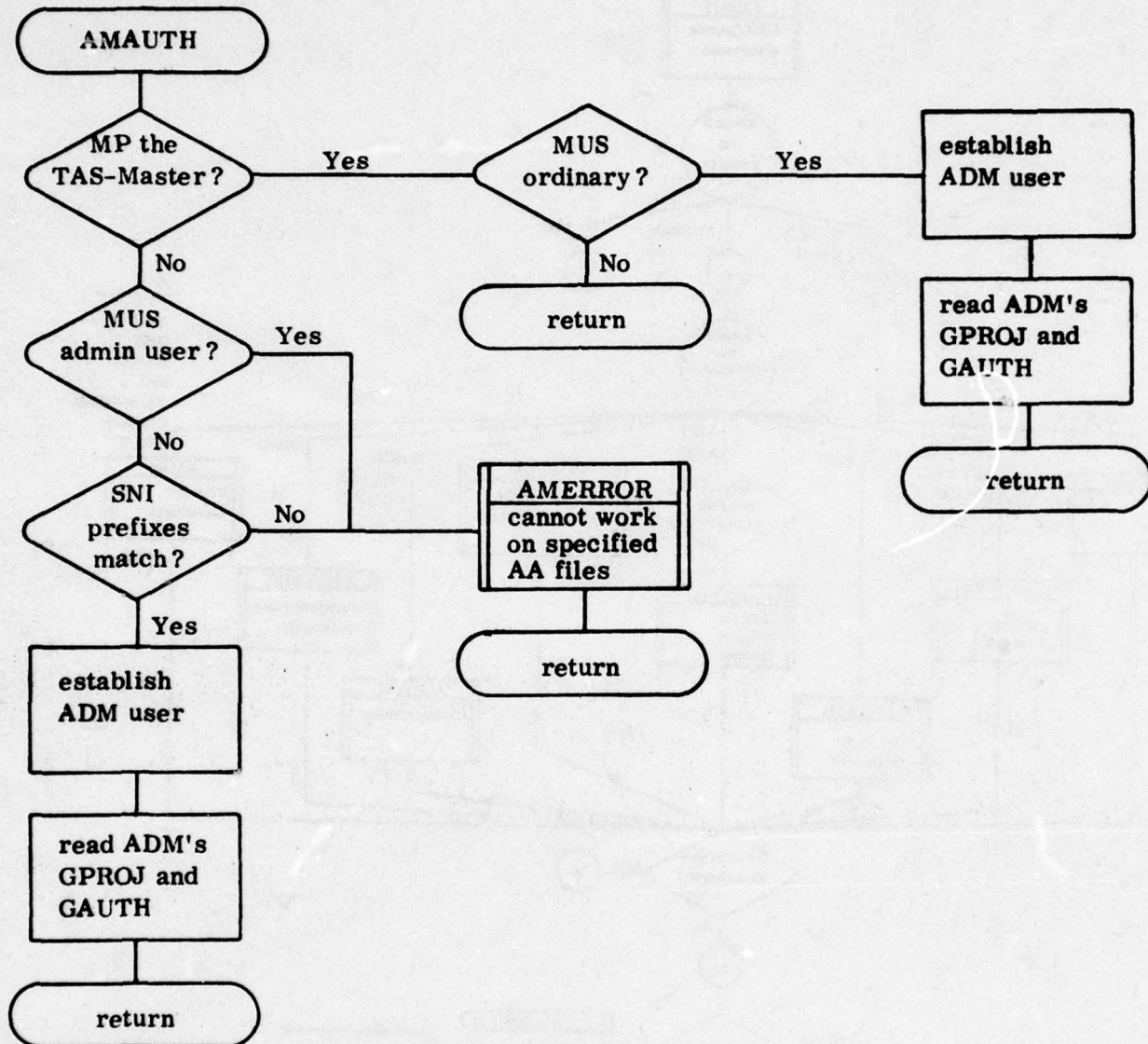


Figure 8. AAFM Process Data Flow (Sheet 11 of 32)

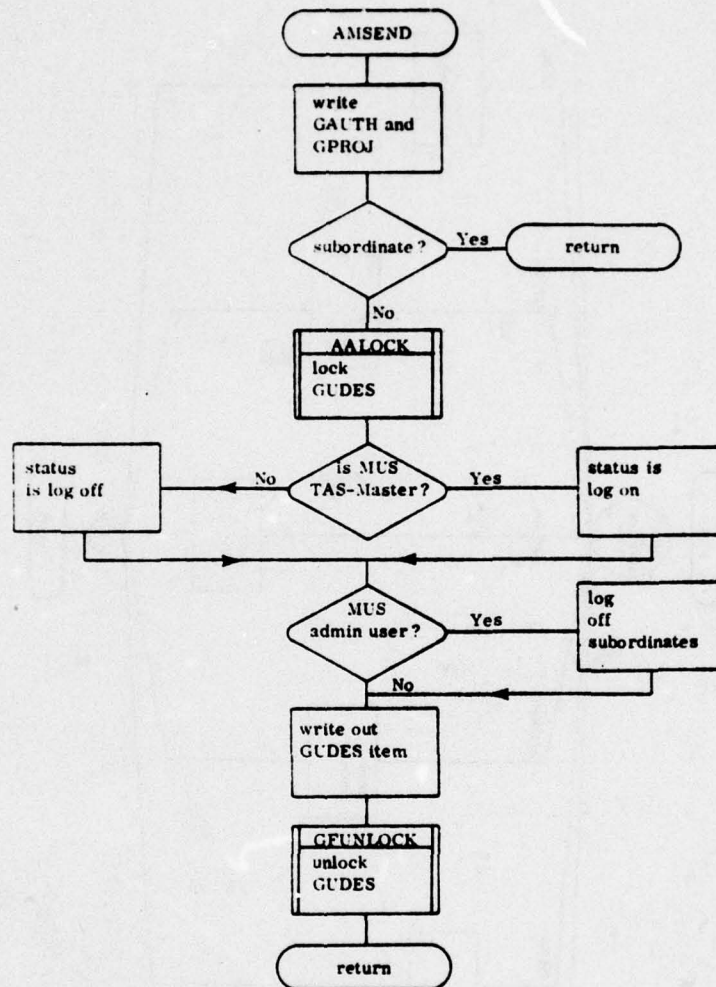


Figure 8. AAFM Process Data Flow (Sheet 12 of 32)

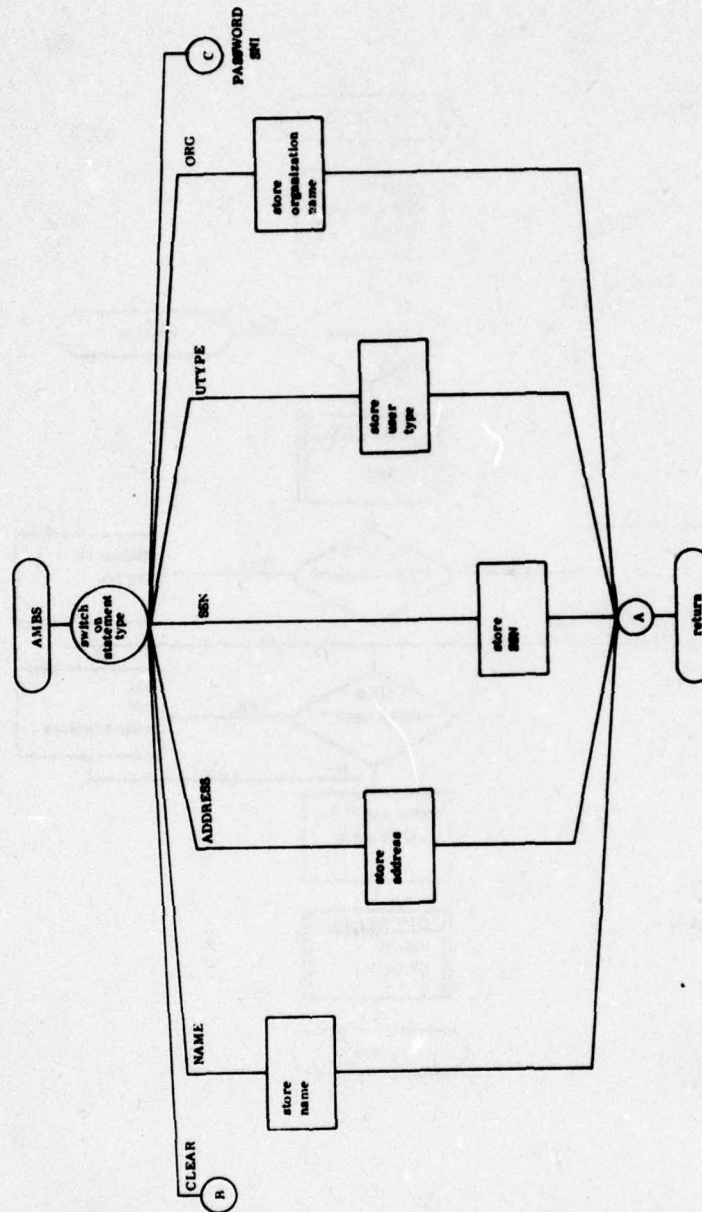


Figure 8. AAFM Process Data Flow (Sheet 13 of 32)

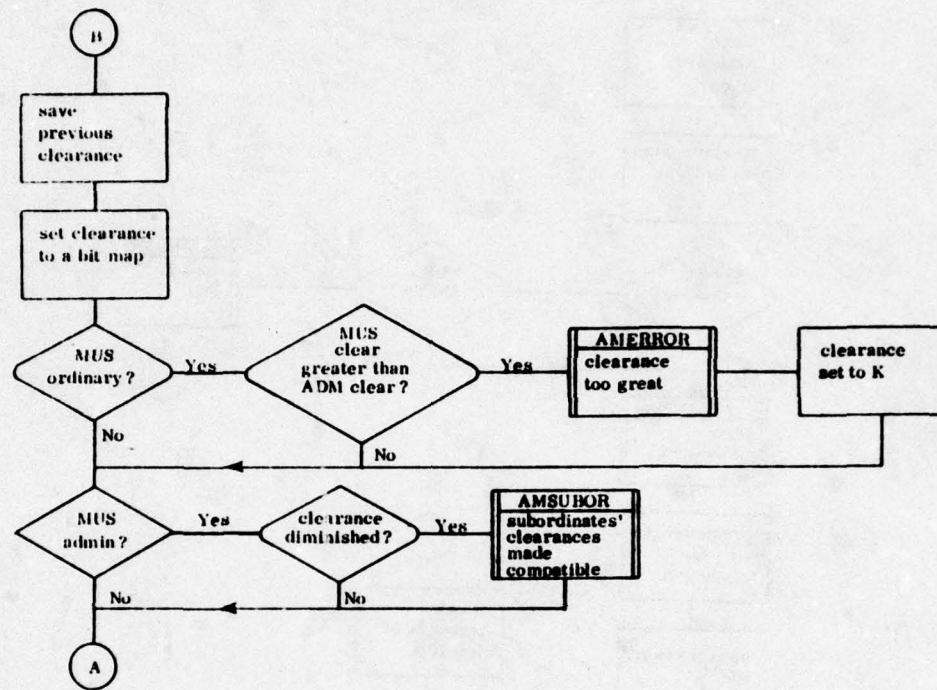


Figure 8. AAFM Process Data Flow (Sheet 14 of 32)

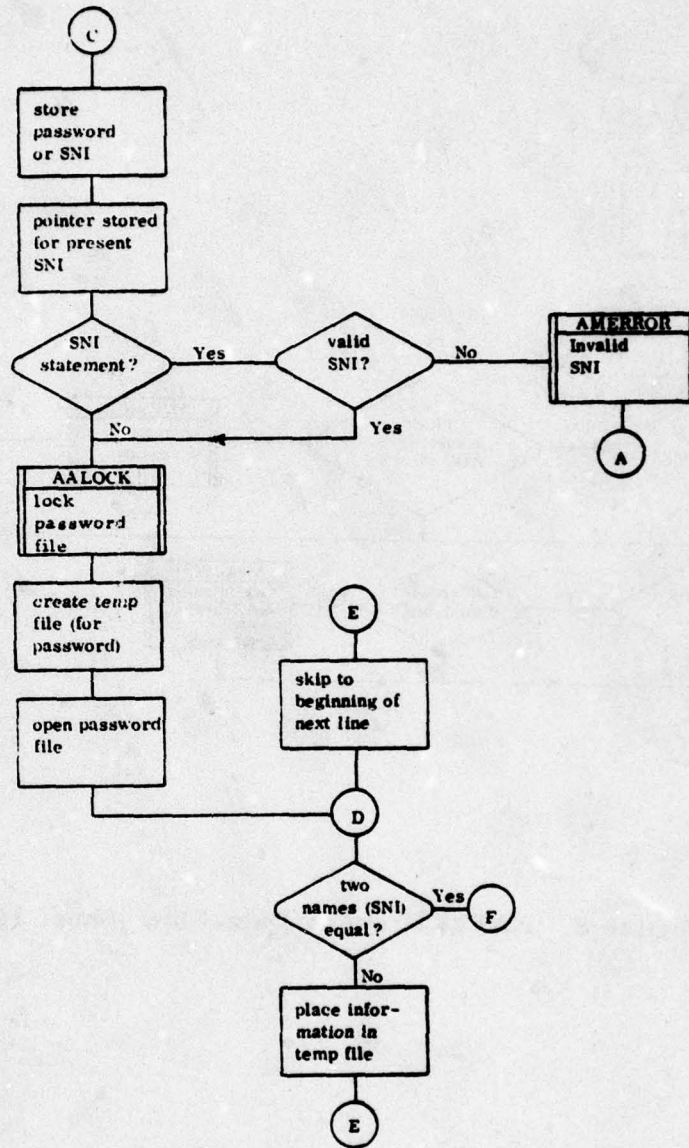


Figure 8. AAFM Process Data Flow (Sheet 15 of 32)

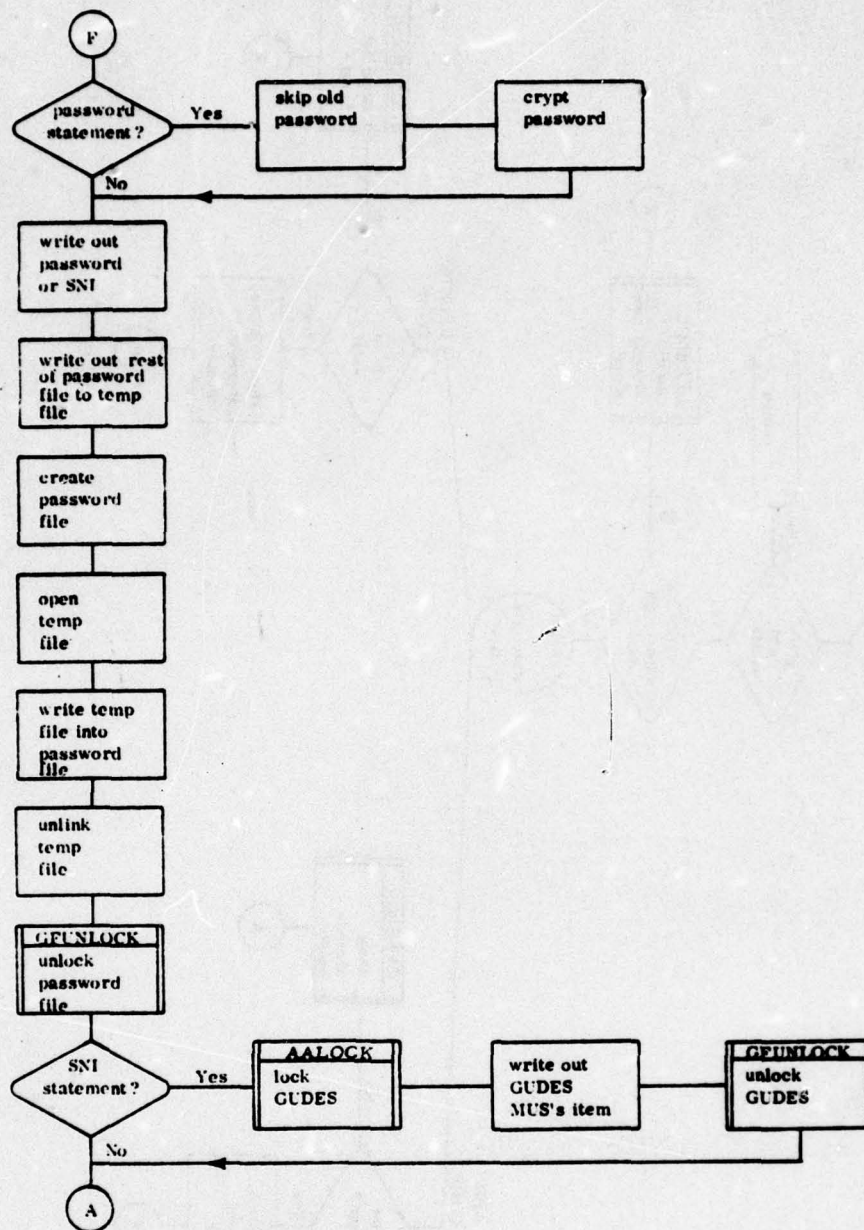


Figure 8. AAFM Process Data Flow (Sheet 16 of 32)

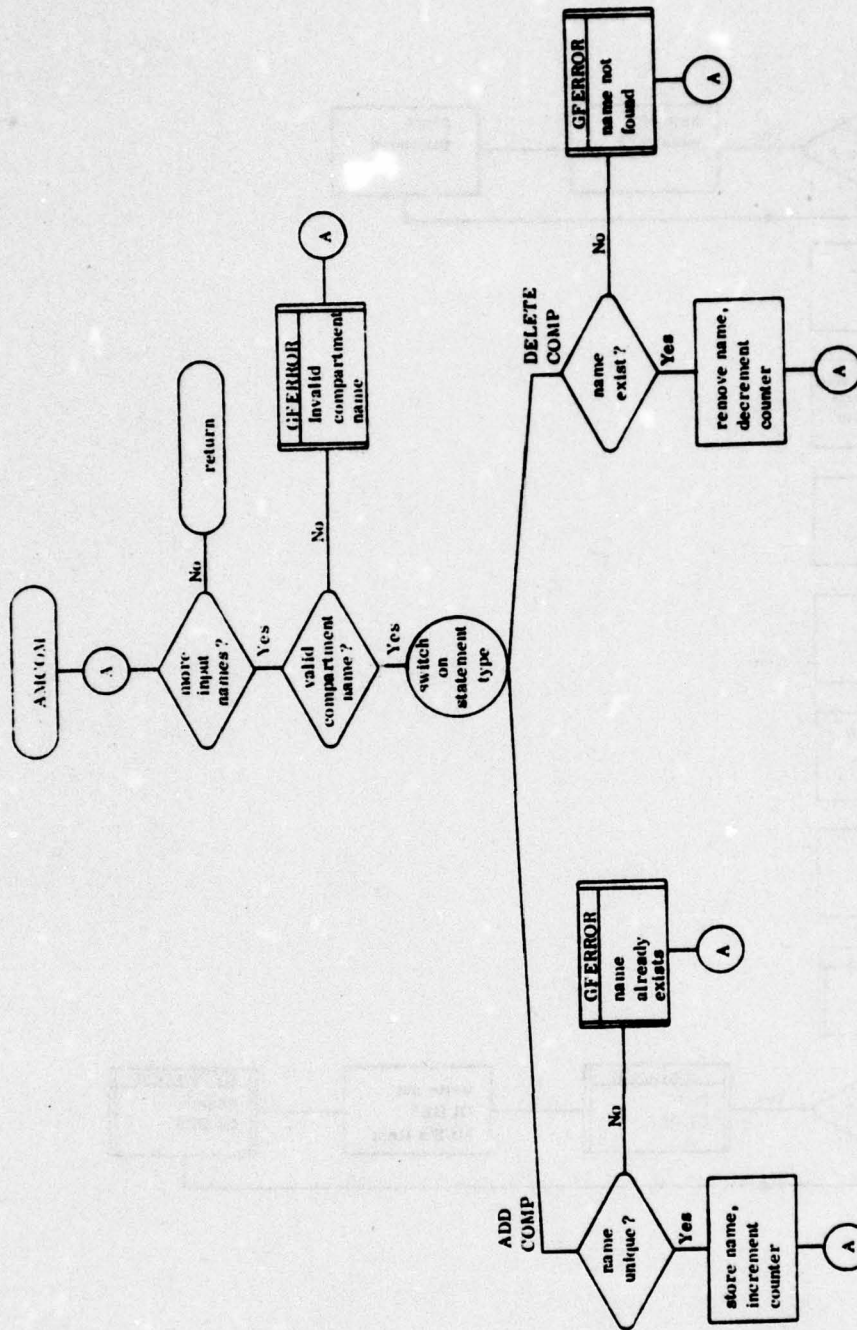


Figure 8. AAFM Process Data Flow (Sheet 17 of 32)

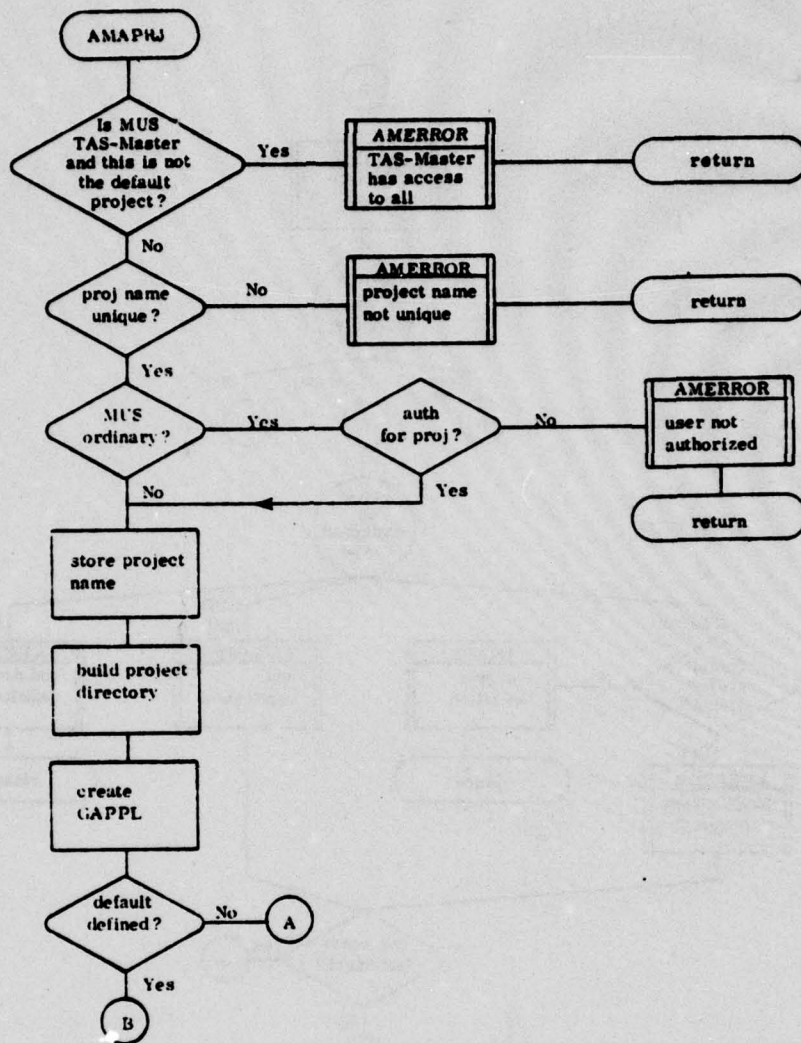


Figure 8. AAFM Process Data Flow (Sheet 18 of 32)

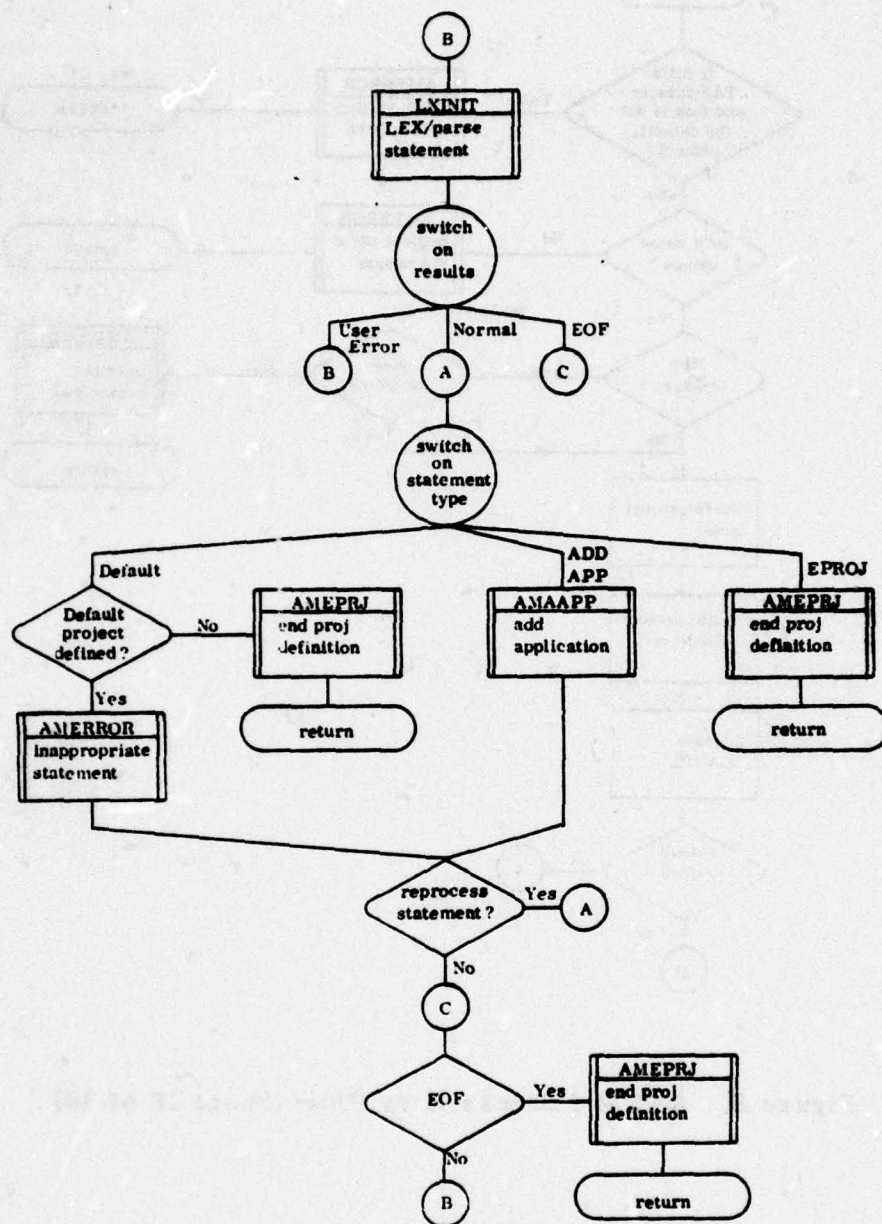


Figure 8. AAFM Process Data Flow (Sheet 19 of 32)

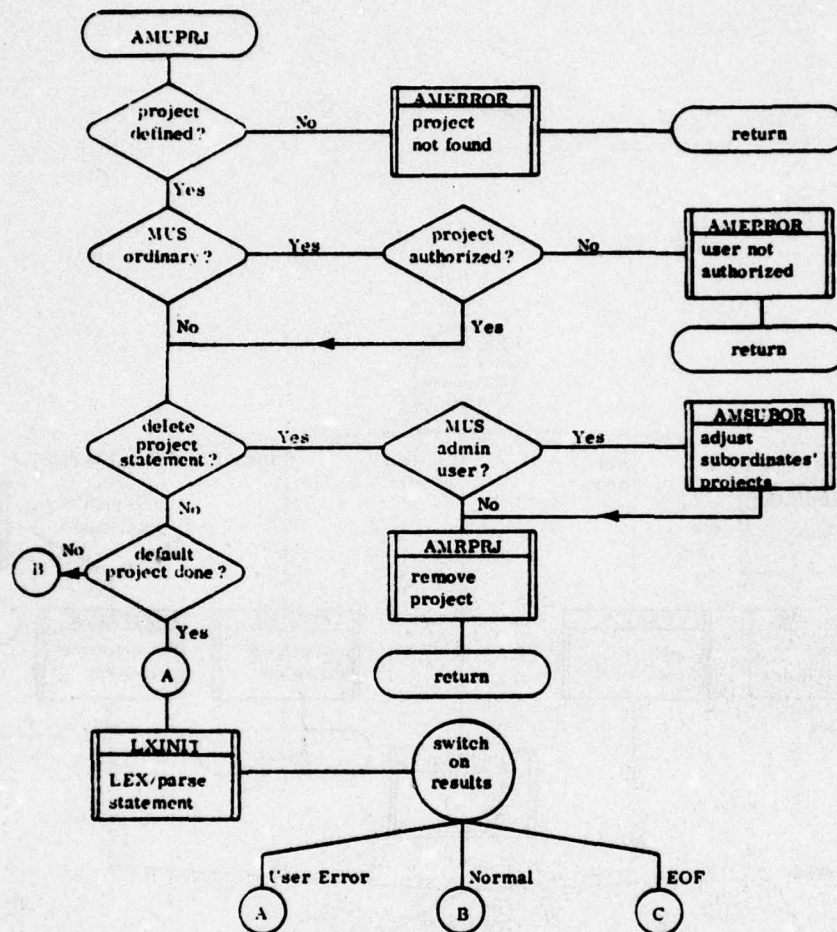


Figure 8. AAFM Process Data Flow (Sheet 20 of 32)

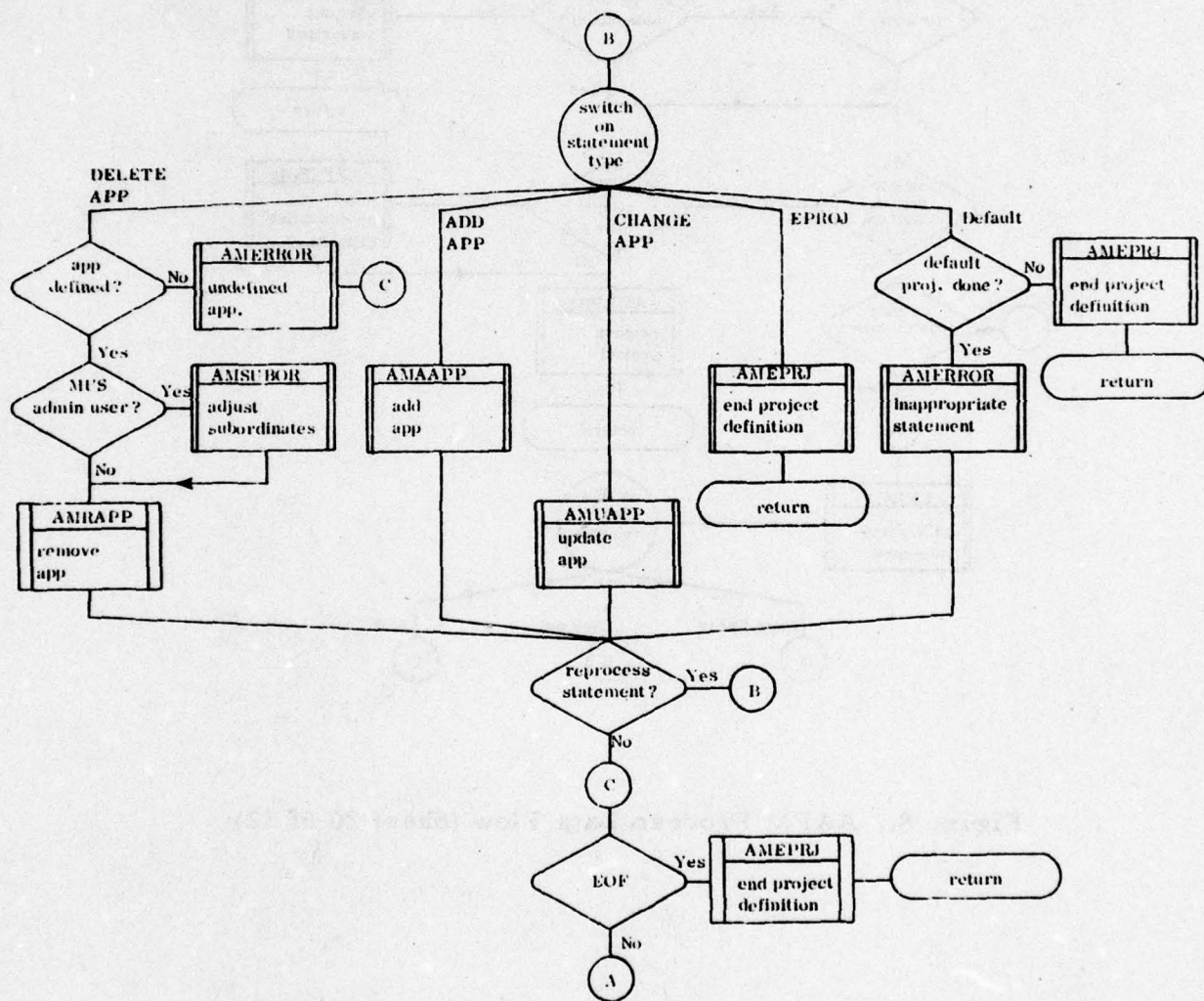


Figure 8. AAFM Process Data Flow (Sheet 21 of 32)

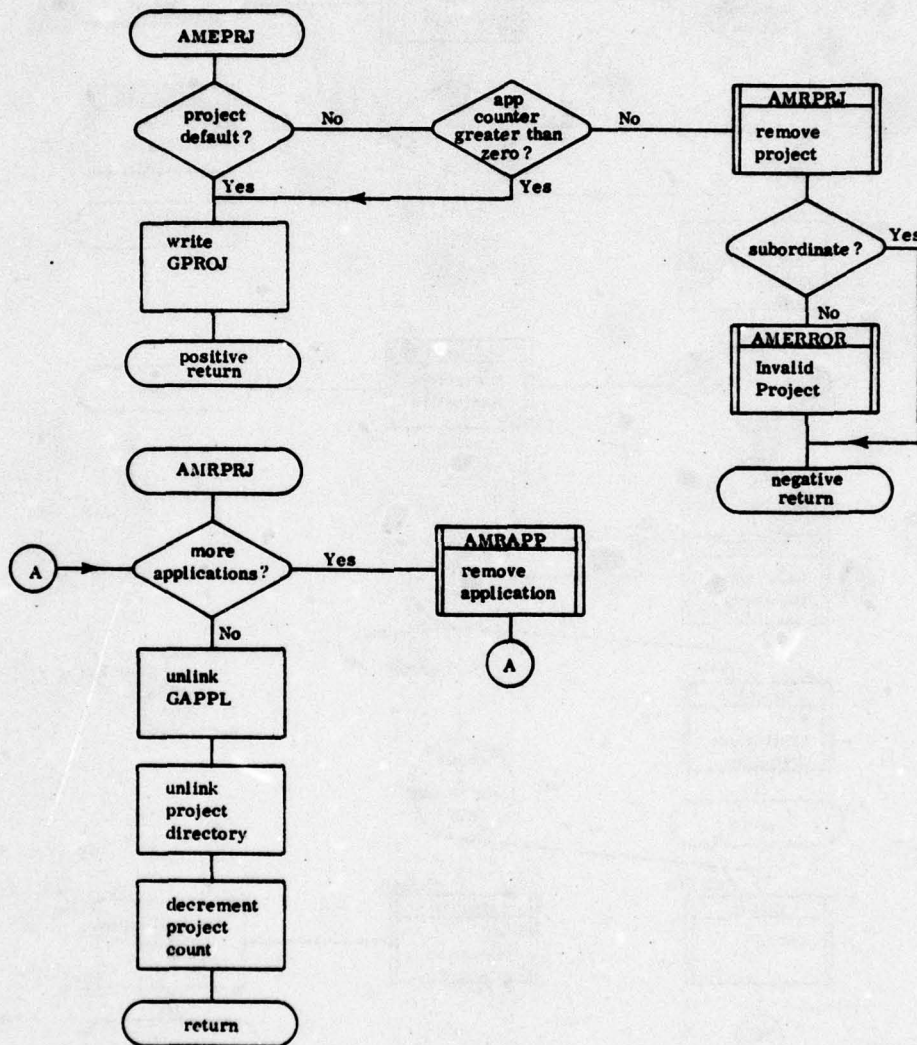


Figure 8. AAFM Process Data Flow (Sheet 22 of 32)

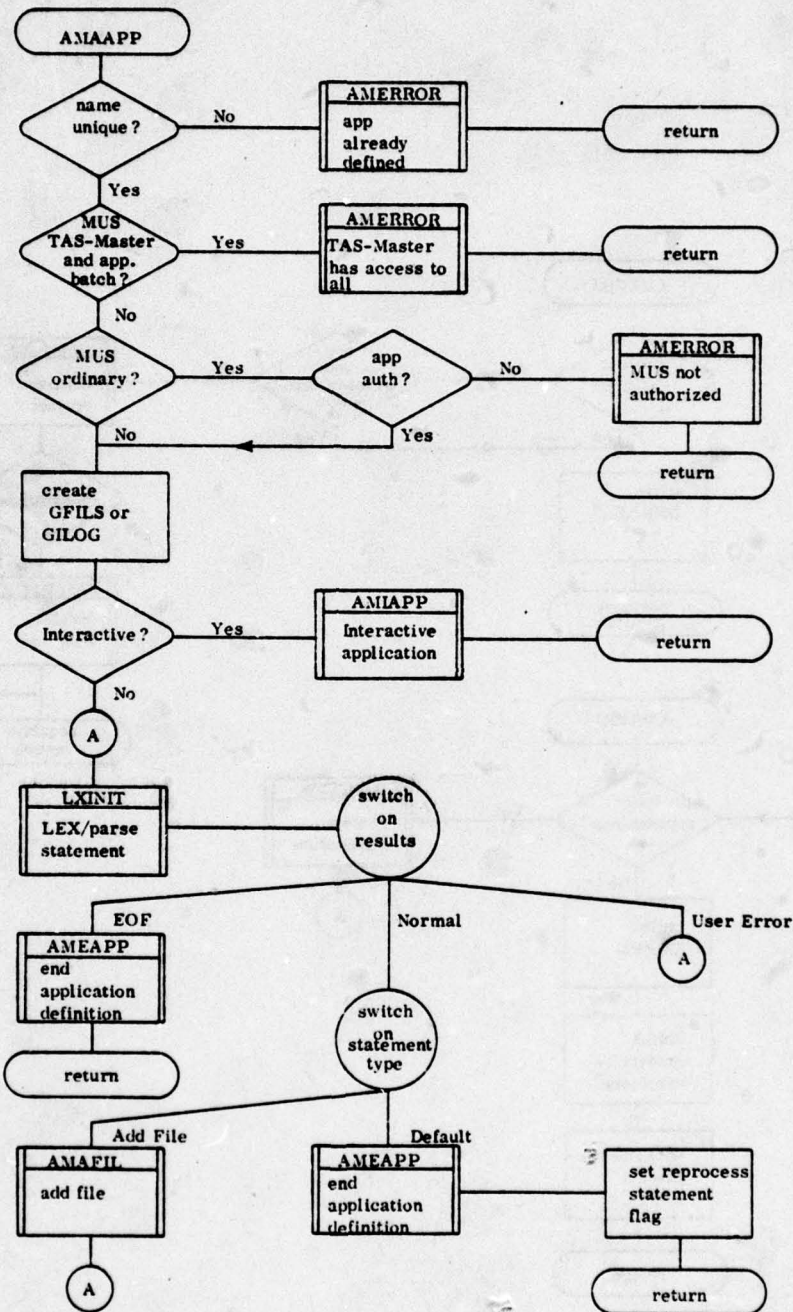


Figure 8. AAFM Process Data Flow (Sheet 23 of 32)

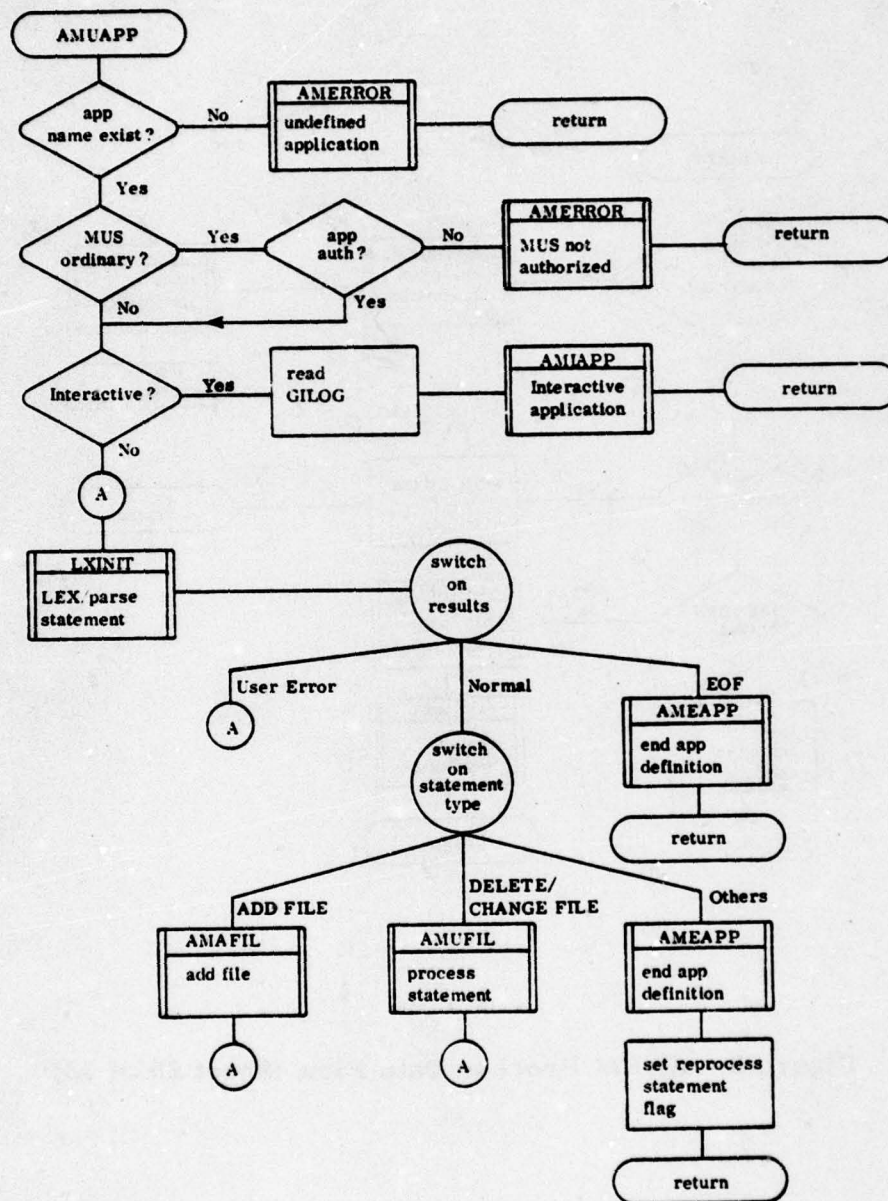


Figure 8. AAFM Process Data Flow (Sheet 24 of 32)

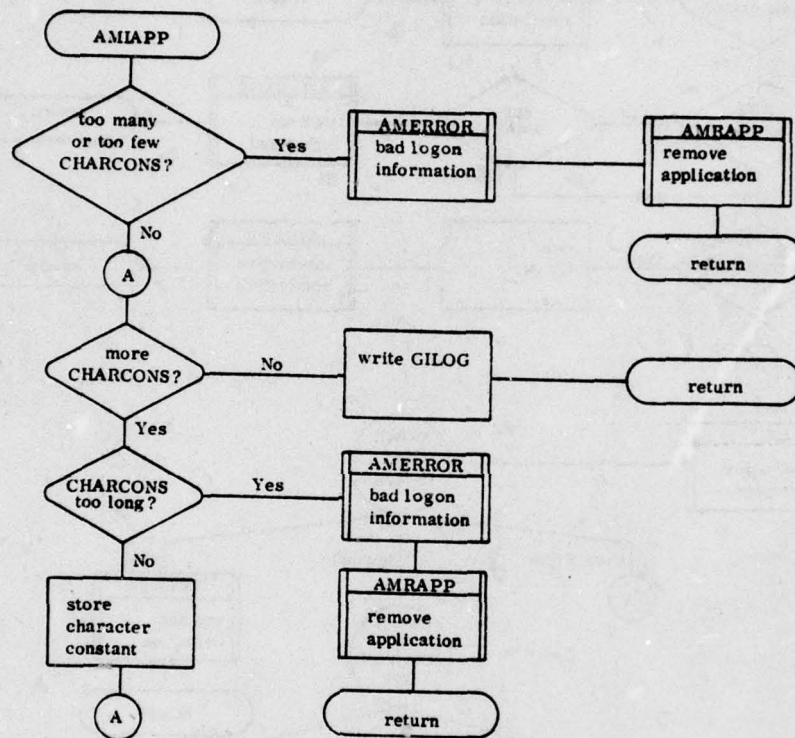


Figure 8. AAFM Process Data Flow (Sheet 25 of 32)

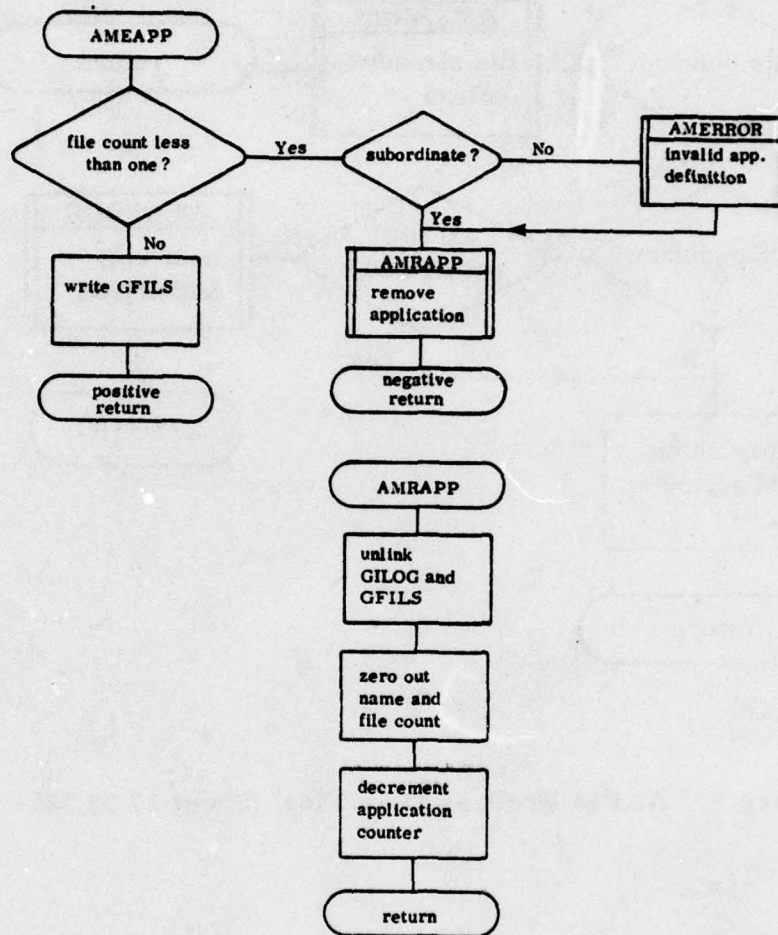


Figure 8. AAFM Process Data Flow (Sheet 26 of 32)

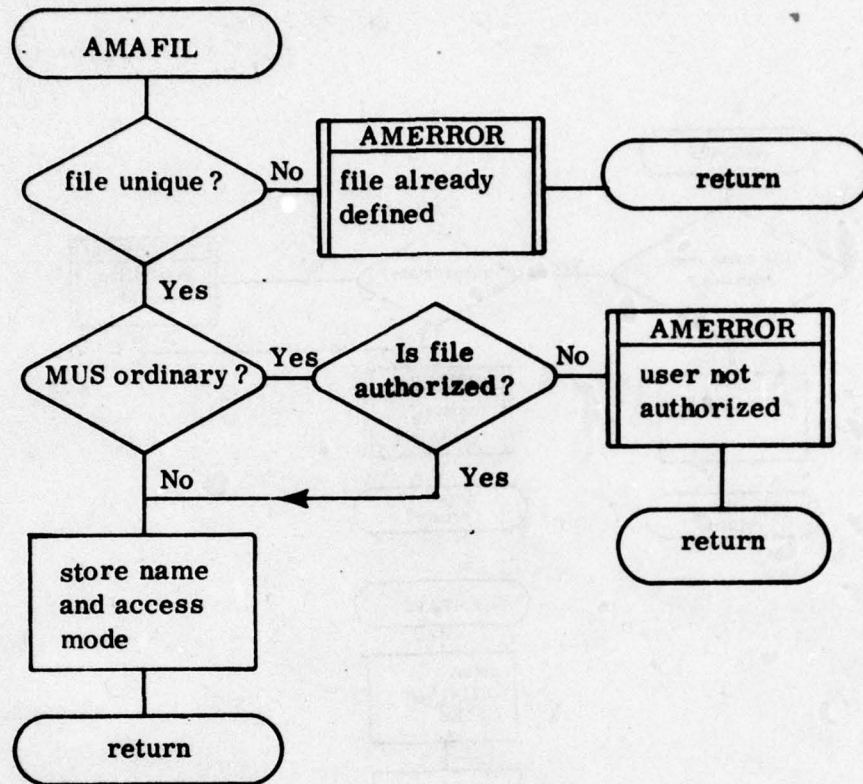


Figure 8. AAFM Process Data Flow (Sheet 27 of 32)

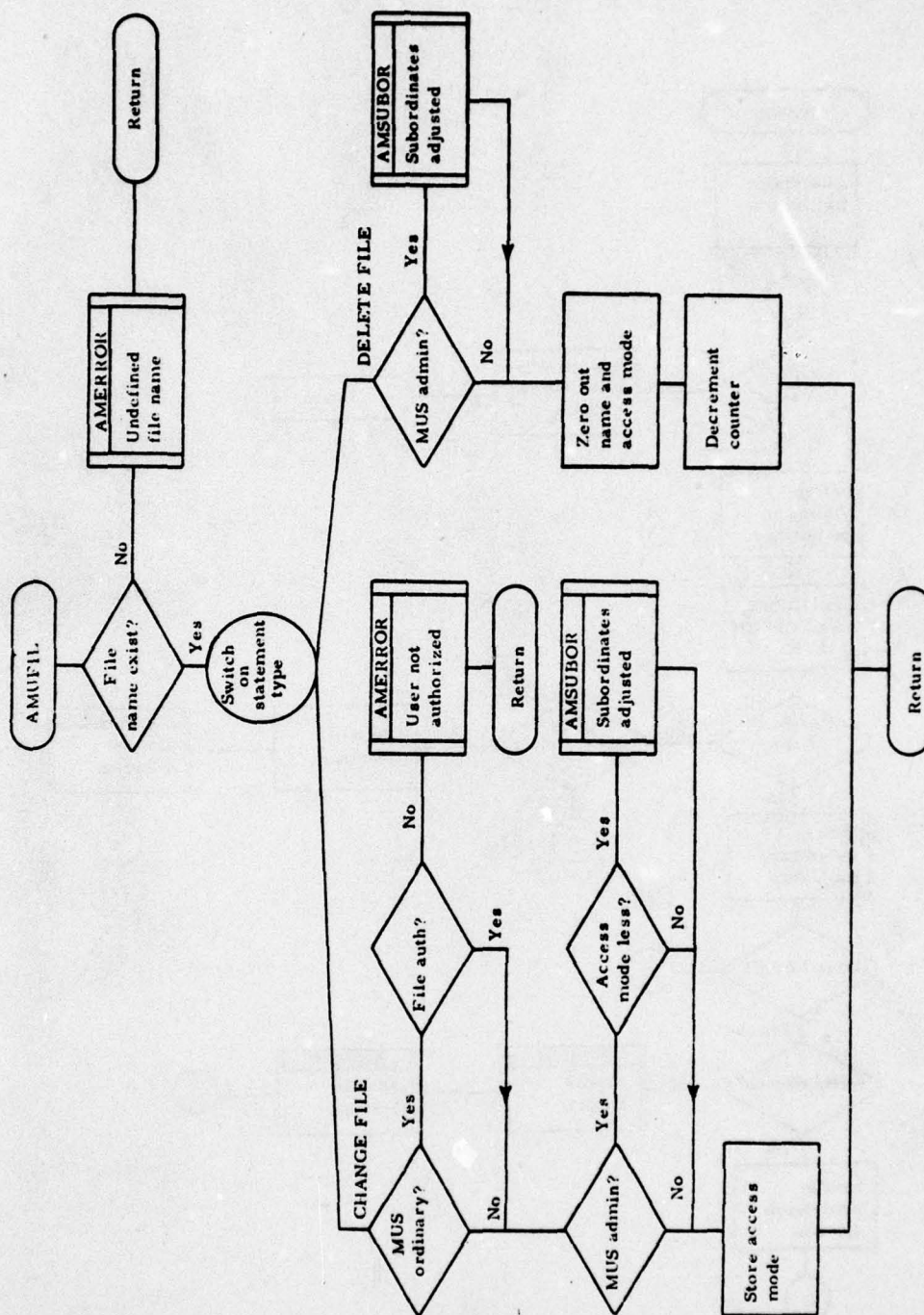


Figure 8. AAFM Process Data Flow (Sheet 28 of 32)

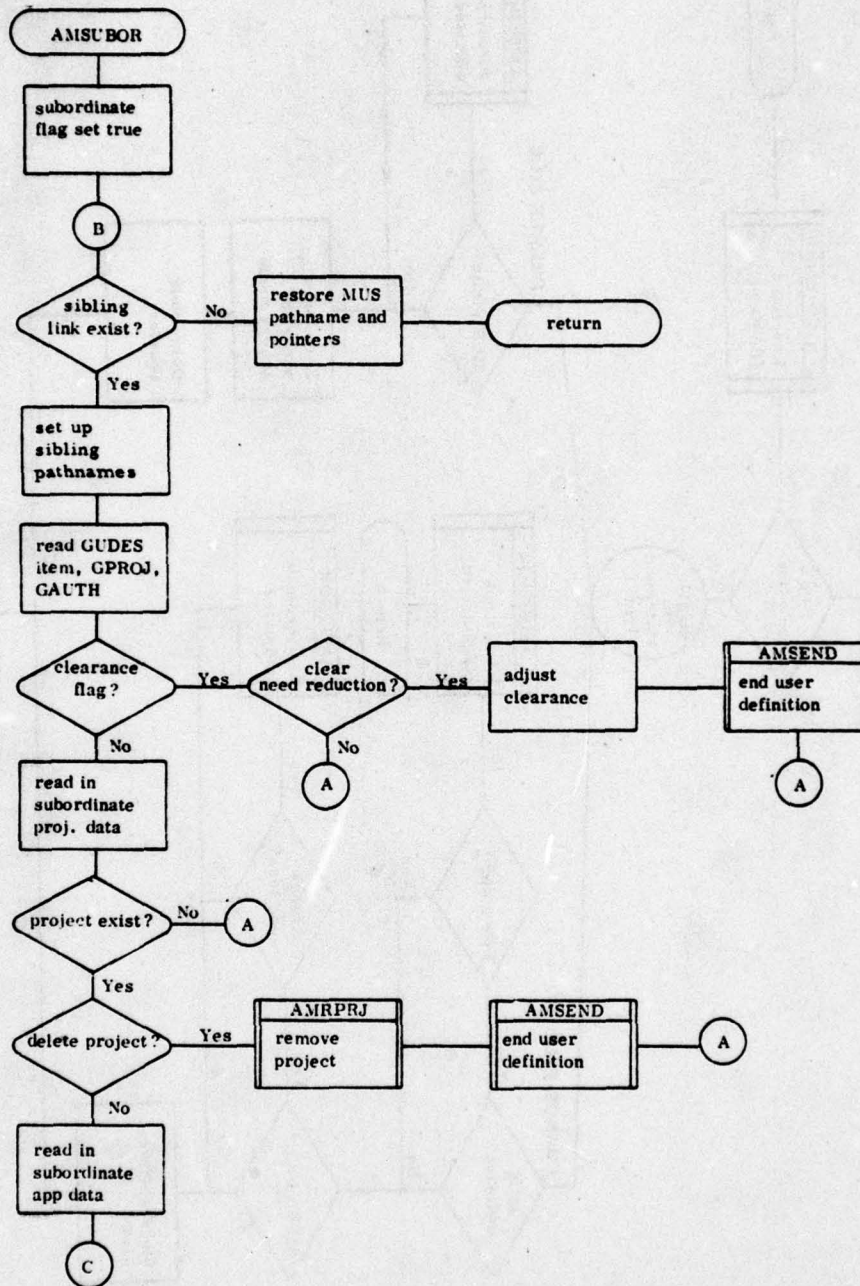


Figure 8. AAFM Process Data Flow (Sheet 29 of 32)

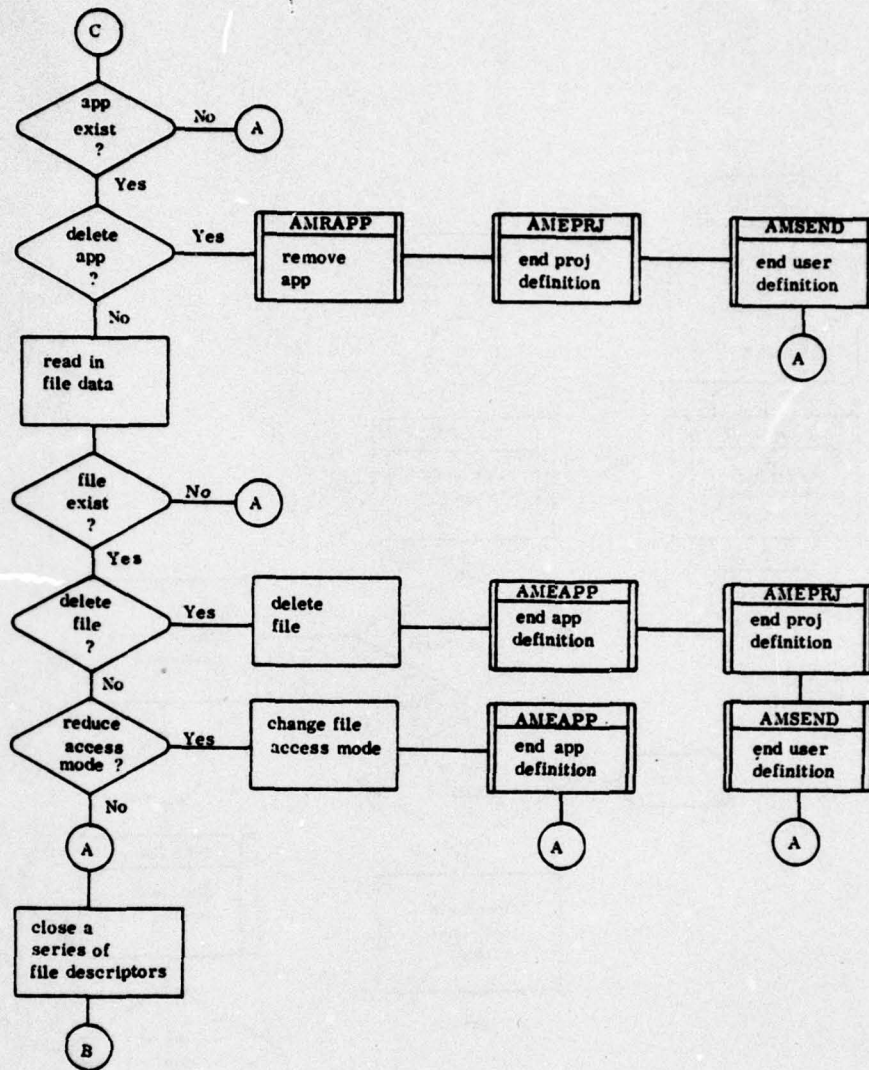


Figure 8. AAFM Process Data Flow (Sheet 30 of 32)

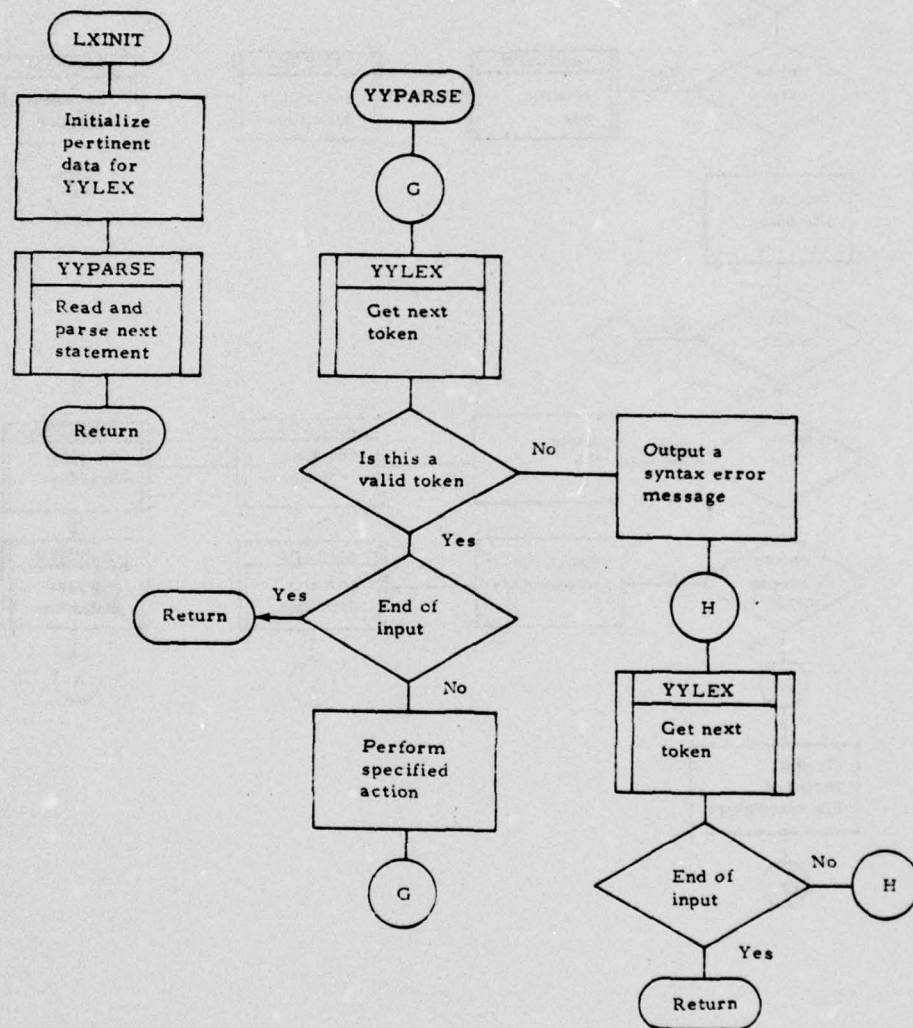


Figure 8. AAFM Process Data Flow (Sheet 31 of 32)

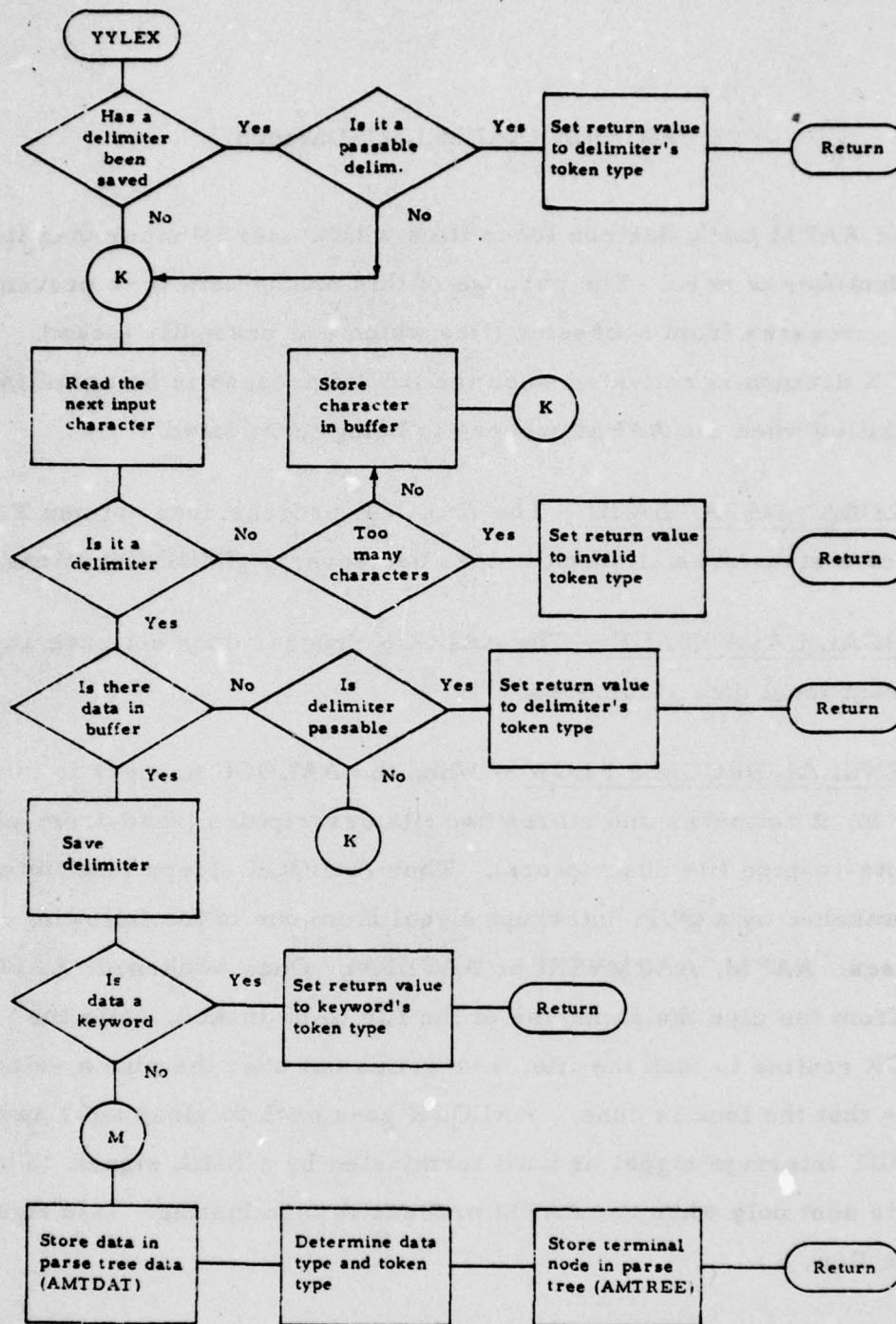


Figure 8. AAFM Process Data Flow (Sheet 32 of 32)

AALOCK (AAFM Lock Daemon)

The AAFM Lock Daemon locks files with a user ID other than the super-user identifier of zero. The purpose of this mechanism is to prevent the AAFM processes from accessing files which are presently locked. The AALOCK daemon is activated when the AAFM process is being initiated and is killed when the AAFM process is being terminated.

GLOBAL DATA USAGE - The AALOCK process does not use TAS global data structures although it does use several global definitions.

LOCAL DATA USAGE - The AALOCK process does not have any significant local data structures.

GENERAL PROCESS FLOW - When the AALOCK process is initiated by AAFM, it retrieves and stores two file descriptors (read-from-pipe and write-to-pipe file descriptors). Then AALOCK sleeps indefinitely until awakened by a QUIT interrupt signal from one of the following AAFM processes: AAFM, AARMVSN or AATERM. Once awakened, AALOCK reads from the pipe the pathname of the file to be locked, calls the GFLOCK routine to lock the file, and writes out over the pipe a value to indicate that the lock is done. AALOCK goes back to sleep until awakened by a QUIT interrupt signal or until terminated by a KILL signal. The latter signal is sent only when the AAFM process is terminating. (See figure 9 for data flow.)

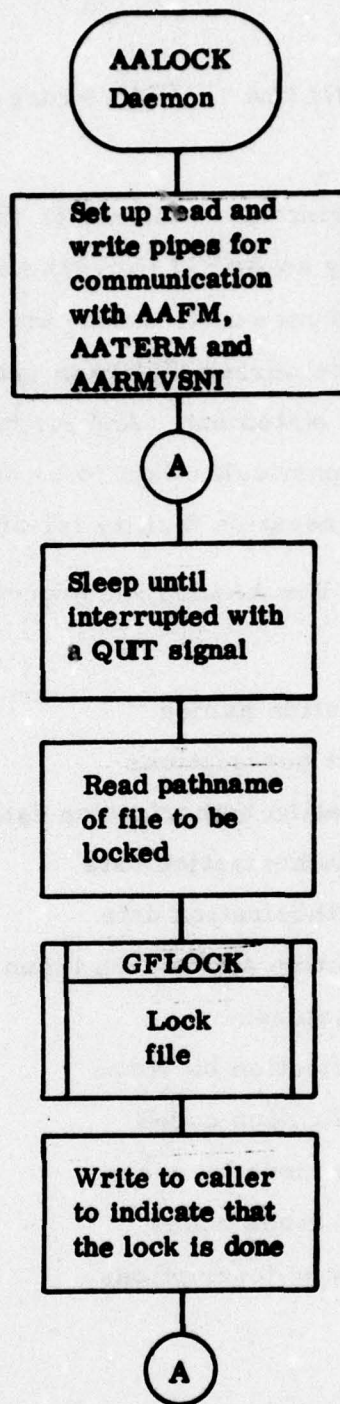


Figure 9. AALOCK Process Data Flow

AALISTSNI (AA List SNI Process)

The AALISTSNI process prints the contents of TAS users' access authorization (AA) files as specified by an AAFM list-sni statement. The AAFM process calls AALISTSNI to process a statement which is stored in AMTREE and AMTDAT. The maintenance person (MP) can specify a TAS user or an agency-group in the list-sni statement. And further, the MP can request all data or specify the particular data to be output. This essentially provides a simplistic report generation facility for an administrative user.

GLOBAL DATA USAGE - The AALISTSNI process uses the following global data structures.

GAPNM	-	application names
GPROJ	-	project descriptions
GAPPL	-	application authorization data
GAUTH	-	user authorization data
GFILS	-	file authorization data
GILOG	-	interactive application logon information
GHOST	-	host statuses
GMAP	-	classification bit maps
GCLAS	-	classification codes
GAMAP	-	access mode bit maps
GACCM	-	access mode codes
GUTY	-	user type descriptions

LOCAL DATA USAGE - The significant local data variables used by AALISTSNI are:

- AMMPID - The user ID of the Maintenance Person (MP) who is using the AAFM language to request the output.
- AMBASIC - A flag which, if set, indicates that the basic user information is to be output. (Syntactically, this implies that neither an SNI nor a USER LIST is specified in the list-sni statement.)
- AMALL - A flag which, if set, indicates that all access authorization data are to be output. (Syntactically, the list-sni statement has a full SNI specified in the form of the token CHARCONS.)
- AMTREE - AAFM uses the parse tree for AAFM statement processing. In this integer array predictably placed nodes correspond to the pertinent elements in a statement.
- AMTDAT - The AMTDAT character array holds data values for a specific statement. AMTDAT is indexed by pointers stored in the parse tree (AMTREE).

GENERAL PROCESS FLOW - The MP user ID is retrieved from the input arguments sent by AAFM and is stored in AMMPID. Next, the AMTDAT and AMTREE arrays are read in. If there are no user parameters specified in the list-sni statement, the flag AMBASIC is set and the statement is further processed as the list-sni statement when just a USERLIST is processed. A list-sni statement with only a USERLIST as input is handled basically in two ways. If the MP is the TASMASTER, GUES is searched for all administrative users. When such a user is located, the AMLOPT function is called to list the requested user's AA data. On the other hand,

if the user is an administrative user, the AMLINK function is called to list the access authorization of the requested agency-group delegated to the administrative user.

If an sni or sni-prefix is input (via a character-constant) by the MP, it is determined whether the MP has authority to specify the sni. If not, a diagnostic informs the MP and the process terminates. If a valid eight character sni is given, the parse tree AMTREE is scanned for a USERLIST token. If the token does not exist, the AMALL flag is set true. Next the sni is validated to be an existing, current sni in the GUDes file. If not, an error message is output and the process exits; but if the sni is valid, the AMLOPT function is called to list the user's AA files.

If the character-constant is a valid sni-prefix and the administrative user for the agency-group represented by the prefix is located in GUDes, the function AMLINK is called to list the agency-group's AA files. If a USERLIST had not been specified, the AMBASIC flag is set, indicating that only the basic personnel data are to be output.

If the character-constant is an invalid sni-prefix, an error message is output and the process AALISTSNI terminates.

Having output the information requested by the MP via the list-sni statement, the AALISTSNI process terminates with a normal status. (See figure 10 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

AMLOPT (List Option Function) - The AMLOPT function is sent the user ID from which it sets up the appropriate pathnames and reads in GAUTH. If the AMBASIC flag is set, the user's sni, name, and organization are output and the function returns to the calling function; otherwise, just the sni is printed. If AMALL is true the following user data is output in the given order.

1. Name
2. Clearance
3. Organization
4. Address
5. Telephone
6. Social Security Number
7. User type
8. Number of Compartments
9. Compartment Names
10. Projects
11. Applications per Project
12. Interactive Logon Information/Batch Files and Access Modes

If the MP specified output via a **USERLIST**, the tokens stored in the list are processed and the data are output in the order in which the **USERLIST** was input and built. A switch is made on each token type and the appropriate information is output. Name, organization, address, telephone number, social security number, and compartment names are personnel information stored as character arrays in **GAUTH**. These are simply transferred into a buffer and output. The user's clearance bit map, stored in **GAUTH.GUCLR**, has to be matched to a bit map in **GMAP**. The index for **GMAP** is used to index **GCLAS** which contains the descriptive mnemonics for the classification bit maps. This clearance information is output. The user type, stored in **GAUTH.GUTYP**, is a numeric index for the **GUTY** file which contains the actual words; **STUDENT**, **CRISIS** and **OPER**.

AD-A050 967

LOGICON INC SAN DIEGO CALIF
TERMINAL ACCESS SYSTEM (TAS) SYSTEM DESIGN SPECIFICATION.(U)
JAN 78 R M BARNHART, L R ERICKSON

F/G 9/2

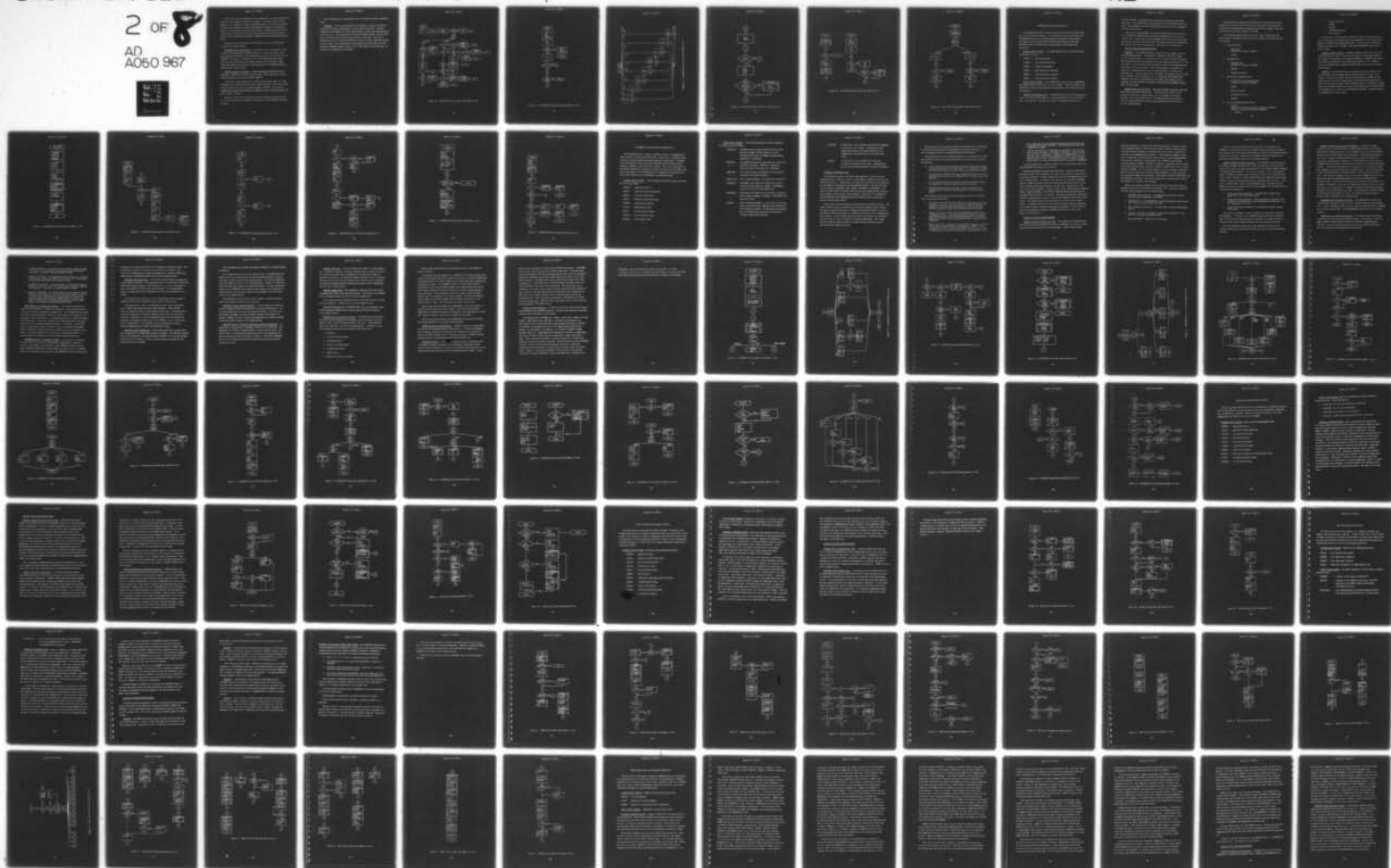
N00014-76-C-0899

UNCLASSIFIED

76-C-0899-6

NL

2 OF 8
AD
A050 967



When a user specifies PROJECT in the USERLIST, just the user's project names, as they appear in GPROJ, are output. If the token APP or FILE appears in the USERLIST, function AMLOPT prints the user's project names and then calls AMWAPP to output each application defined for the present project. (Token FILE causes all project names, all applications per project and all batch-files/interactive-logon data per application per project to be output. The combination of AMLOPT, AMWAPP and AMWFILE accomplishes this.)

After processing each token in the USERLIST, the function AMLOPT returns with a normal status.

AMWAPP (Write Application) – The ASCII version of the project index is inserted into pathnames enabling GAPPL to be opened and read. Each non-empty slot of GAPPL causes the application name (via table look-up into GAPNM) to be output. If AMALL is set or if the USERLIST token type is FILE, the function AMWFIL is called to output further host data. After listed each application defined for the given project, the AMWAPP function returns with a normal status.

AMWFILE (Write File Data) – After inserting the ASCII host ID into pathnames, AMWFIL switches on the host type, either interactive or batch. This information is provided by GHOST.GHTYP.

If the host is a batch application, GFILS is opened and read. For each non-empty slot, the filename and access mode is printed. The access mode of the file maps onto an access mode bit map in GAMAP. The index for GAMAP also indexes GACCM which contains the output representations of the access mode bit maps.

However, if the host is an interactive host, GILOG is opened and read. For each non-empty slot of GILOG, the character-constant logon information is output.

After having output the application's access authorization data, AMWFIL exits.

AMLINK — The function AMLINK calls AMLOPT to output an administrative user's and his subordinates' access authorization data. Initially, AMLINK calls AMLOPT to list the requested data stored in the administrative user's AA files. By picking up the sibling link GUDS.GLINK, the next member of the administrative user's agency-group is specified. That user's GUDS entry item is read and AMLOPT is called to list the specified contents of his AA files. The sibling chain is sequentially processed until one of the member's GUDS.GLINK is equal to zero, indicating the end of the chain. A return to the calling function is made.

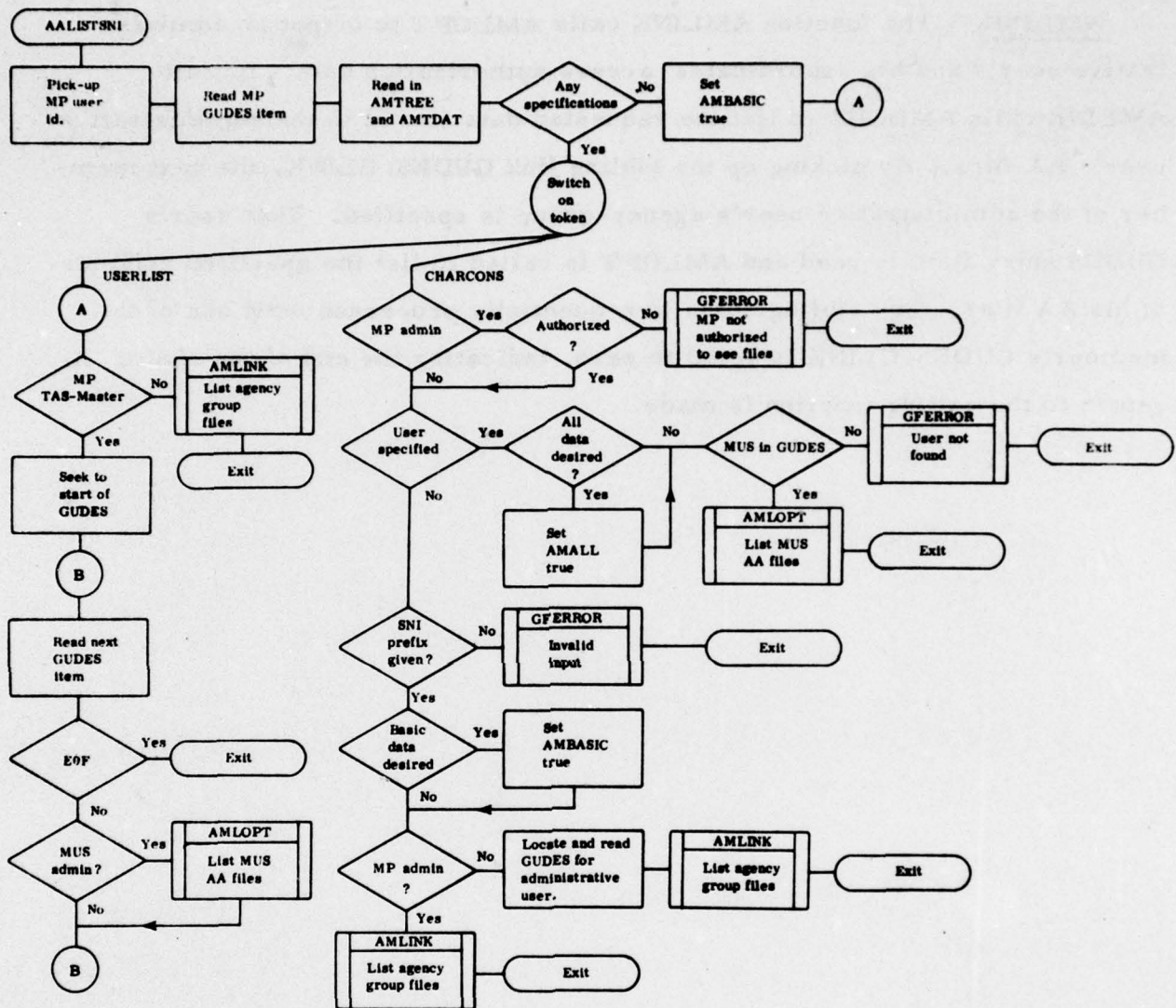


Figure 10. AALIST SNI Process Data Flow (Sheet 1 of 6)

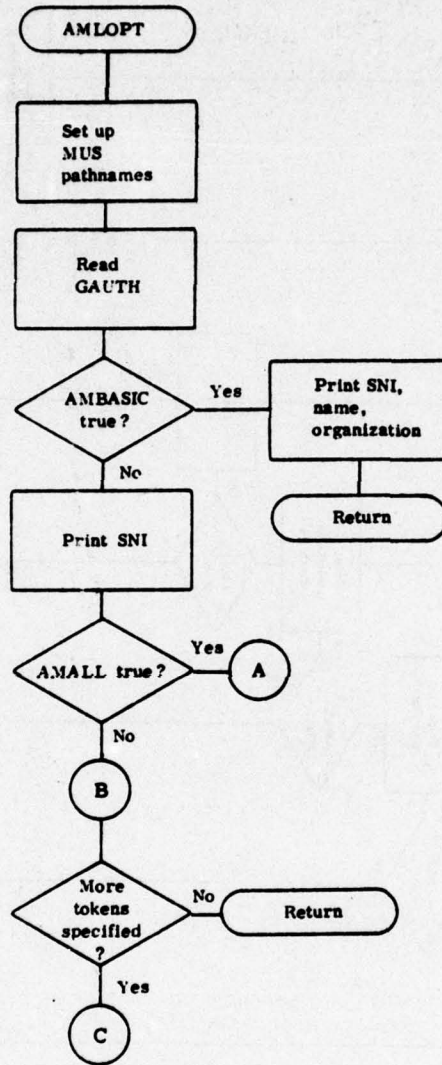


Figure 10. AALISTSNI Process Data Flow (Sheet 2 of 6)

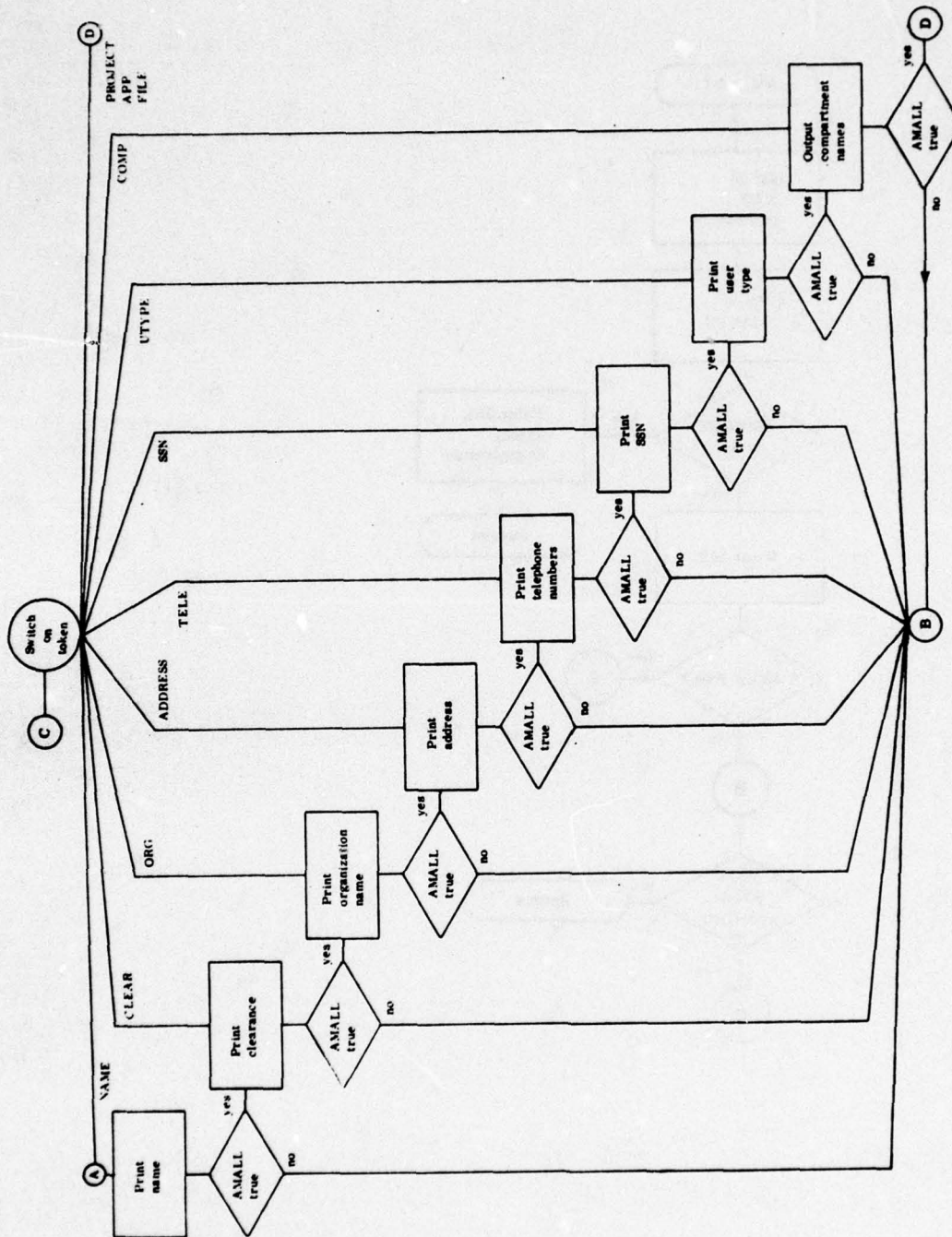


Figure 10. AALISTSN I Process Data Flow (Sheet 3 of 6)

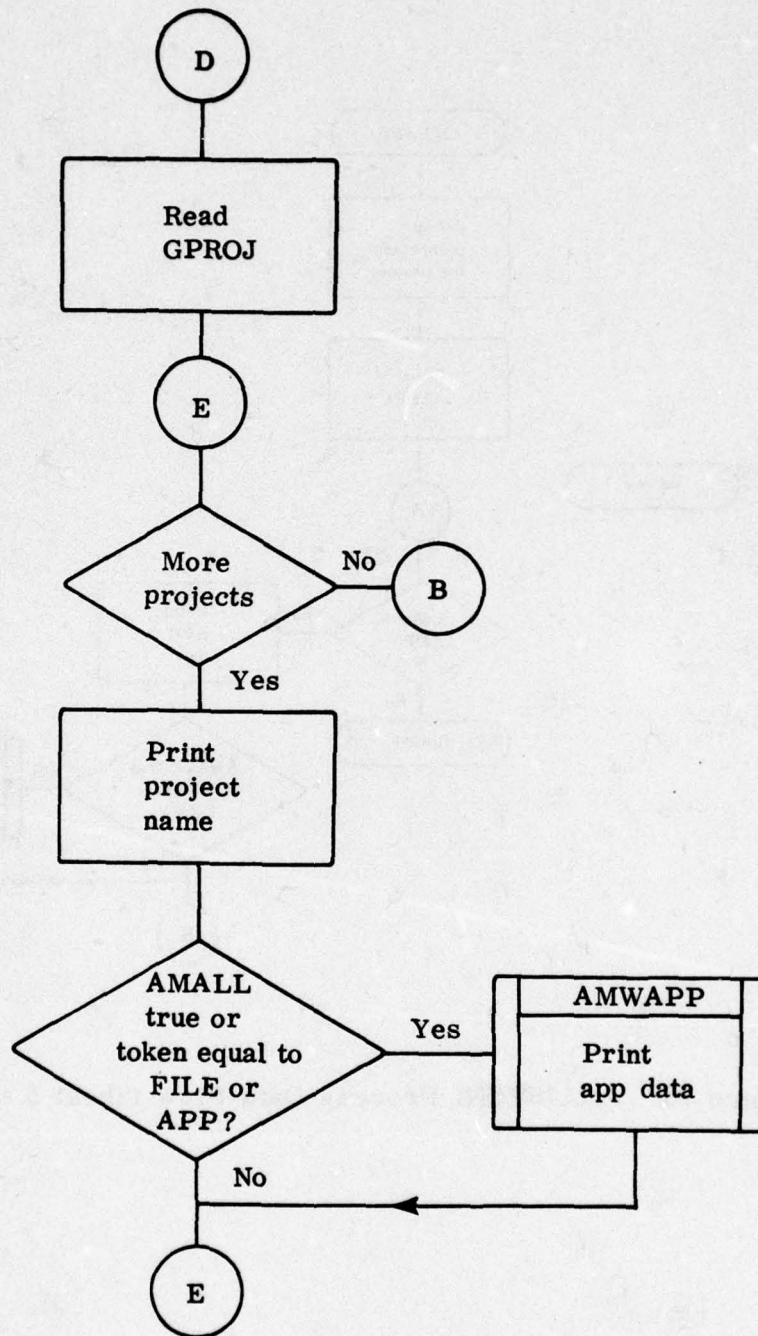


Figure 10. AALISTSNI Process Data Flow (Sheet 4 of 6)

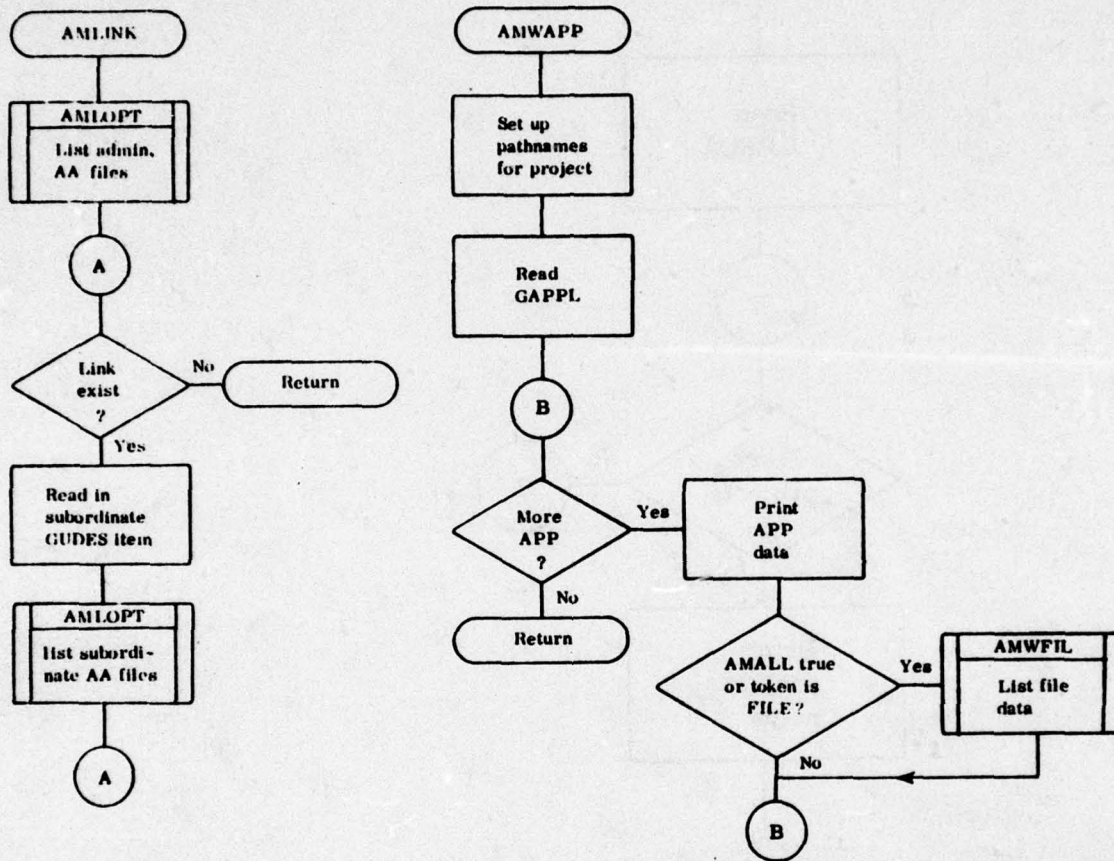


Figure 10. AALIST SNI Process Data Flow (Sheet 5 of 6)

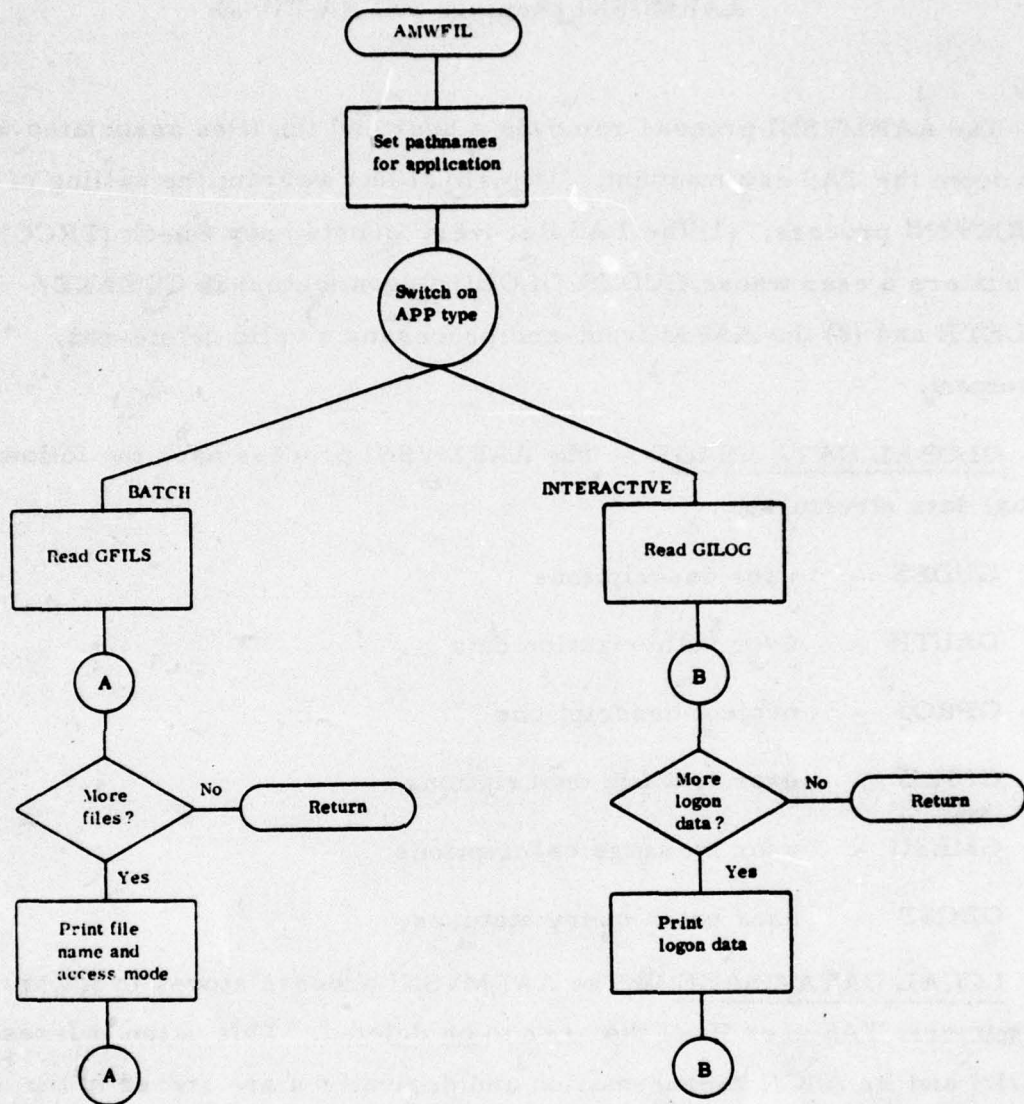


Figure 10. AALISTSNI Process Data Flow (Sheet 6 of 6)

AARMVSNI (Remove SNI AA Files)

The AARMVSNI process removes a user and the files associated with him from the TAS environment. Two situations warrant the calling of the AARMVSNI process: (1) the TAS Recovery Consistency Check (TRCC) encounters a user whose GUDES, GLOGG (logon status) is CREATE/DELETE and (2) the AAFM front-end processes a valid delete-sni statement.

GLOBAL DATA USAGE - The AARMVSNI process uses the following global data structures:

- GUDES - user descriptions
- GAUTH - user authorization data
- GPROJ - project descriptions
- GFILE - user text file descriptions
- GMESG - user message descriptions
- GBQST - user batch query statuses

LOCAL DATA USAGE - The AARMVSNI process stores in AMMUSID the numeric TAS user ID of the user to be deleted. This value indexes into GUDES and its ASCII representation and derivatives are stored in the user file pathnames.

GENERAL PROCESS FLOW - AMRMVSNI stores the three input parameters: the read and write pipe file descriptors and the TAS user ID of the user to be removed. The pipes are used to communicate with the lock

daemon AALOCK, a mechanism by which files are locked by the AAFM processes. The GUDes file is opened and the user's GUDes entry item is read. The AMPASS function is next called to remove the user's logon line from the password file.

Upon return from AMPASS, the user ID (AMMUSID) and derivatives of it are converted to ASCII and are inserted into pathnames for the user's various files. Finally, the function AMRSNI removes the user's files from the TAS file structure and environment. Having accomplished the deletions, the process exits with a normal status. (See figure 11 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

AMPASS - The AMPASS function removes the user's logon line from the UNIX password file, "/etc/passwd". Initially, AALOCK is called to lock the password file. A temporary file is created with which character data manipulation from the password file to the temporary file and vice versa is possible. A search for a match between the snis stored in the password file and the user's sni is made. If the match attempt is successful, the logon line is skipped and the rest of the password file is written to the temporary file. The password file is recreated, the temporary file is closed then reopened, and the temporary file is written onto the password file. Whether or not the sni match effort is successful, the temporary file is unlinked, the password file is unlocked, and the function AMPASS returns with a normal status.

AMRSNI (Remove User Files) - Basically AMRSNI unlinks the files and directories associated with the user. The files and directories can be grouped into four basic classifications: text, message, batch query/response, and access authorization. The following deletion procedure of the text files sufficiently describes the general deletion process for the four classifications.

The GFILE file is opened and the first item is read for a file count. Each item in the file is read; if the entry is non-empty, the filename is inserted into the appropriate pathname and the file is unlinked. Once the end-of-file is processed, GFILE is unlinked.

The following outline indicates the basic order in which files and directories are unlinked in the process of removing a user from the TAS environment.

a. User Text Files

- Text Files
(Names are stored in GFILE)
- GFILE

b. Message Files

- Message Files
(Indexes are stored in GMESG)
- GMESG
- Message directory

c. Batch Query/Response Files

- Batch Query and Response Files
(JOBIDs are stored in GBQST)
- GBQST
- Query Directory
- Response Directory
- GBQFN

d. Access Authorization (AA) Files

- Projects
(Indexes are determined by entry position in GPROJ)
(Removal of projects is done by AMPPRJ)
- GAPPL

- Project Directory
- GPROJ
- GAUTH
- TAS Data Directory
- SNI Directory

After the user's file structure is deleted, a check is made on the user type. If the user is an ordinary user, the sibling link to the deleted user is updated in GUDES by the AMLINK function. The user entry in GUDES is cleared and written out to GUDES. The function AMRSNI returns with a normal status.

AMPPRJ (Delete Project) - AMPPRJ deletes the files associated with a given project for a user. First the ASCII project index and the application indexes are inserted into the project pathnames. AMPPRJ next unlinks files GILOG and GFILS for each application defined for TAS. AMPPRJ returns to AMRSNI which unlinks the GAPPL file and the project directory for the project.

AMLINK - The AMLINK function removes the deleted ordinary user from the agency-group sibling chain of which the user is a link. First GUDES is locked by AALOCK, and then by searching the GUDES file the administrative user for the deleted user is located and the sibling chain is followed until the user whose GUDES. GLINK is equal to AMMUSID is found. That user's GUDES. GLINK is set equal to the deleted user's GUDES. GLINK. The GUDES entry is written out and GUDES is unlocked. AMLINK returns to AMRSNI with a normal status.

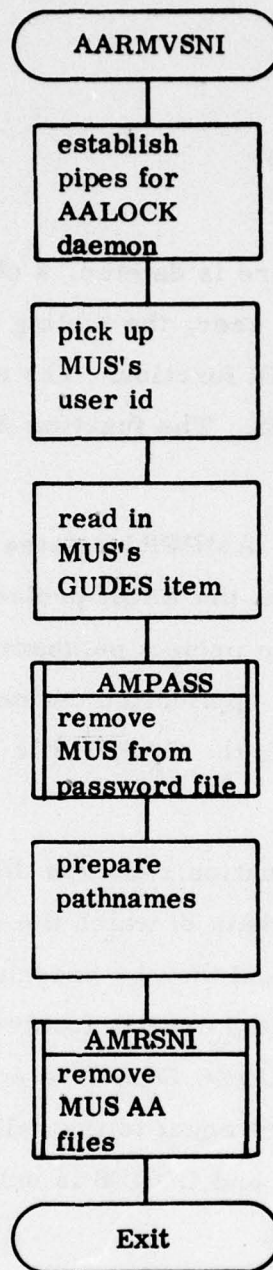


Figure 11. AARMVSNI Process Data Flow (Sheet 1 of 6)

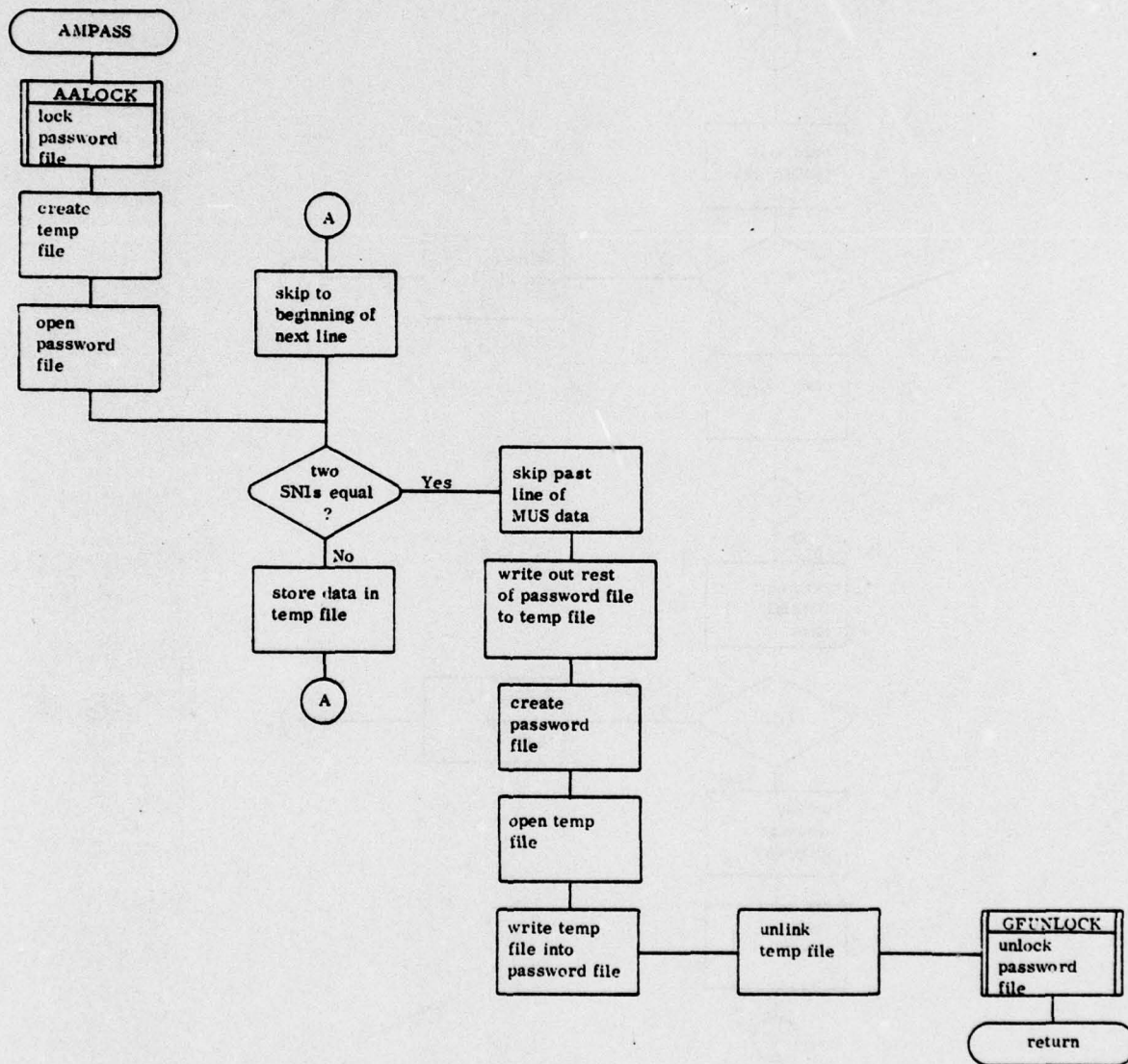


Figure 11. AARMVSNI Process Data Flow (Sheet 2 of 6)

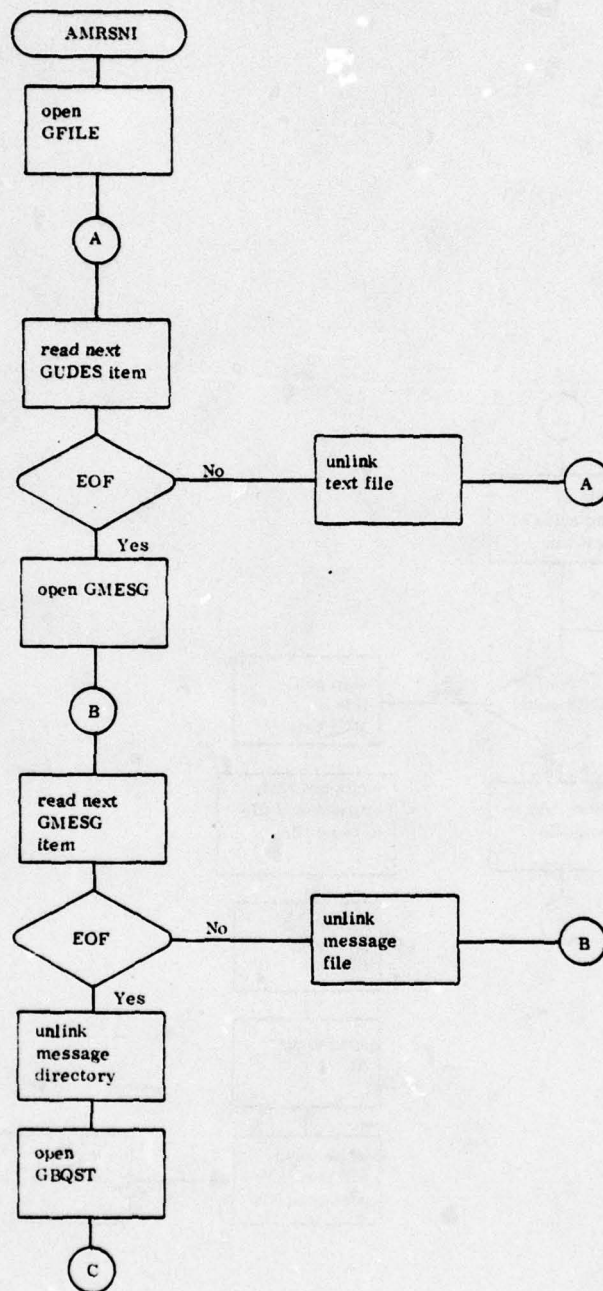


Figure 11. AARMVSNI Process Data Flow (Sheet 3 of 6)

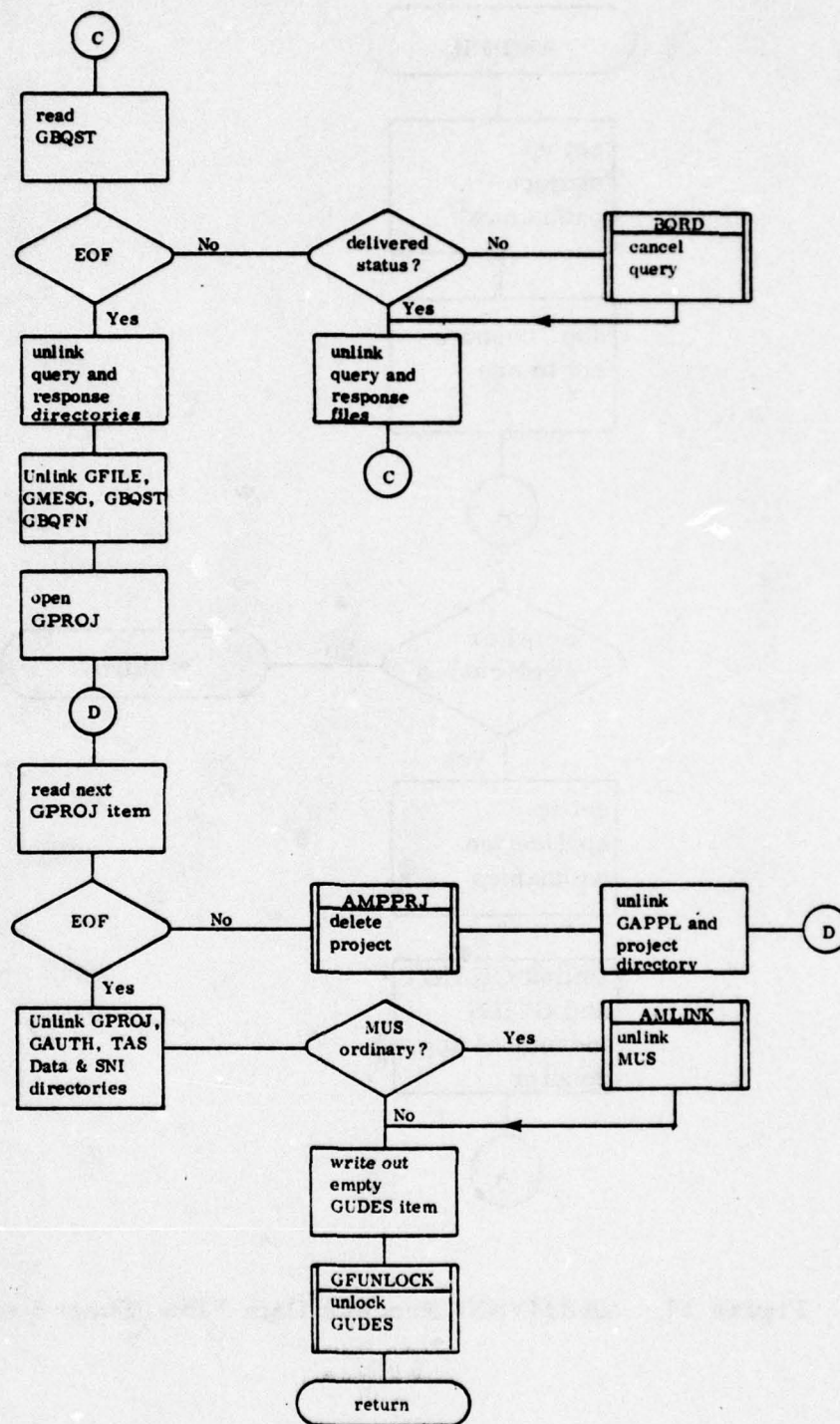


Figure 11. AARMVSNI Process Data Flow (Sheet 4 of 6)

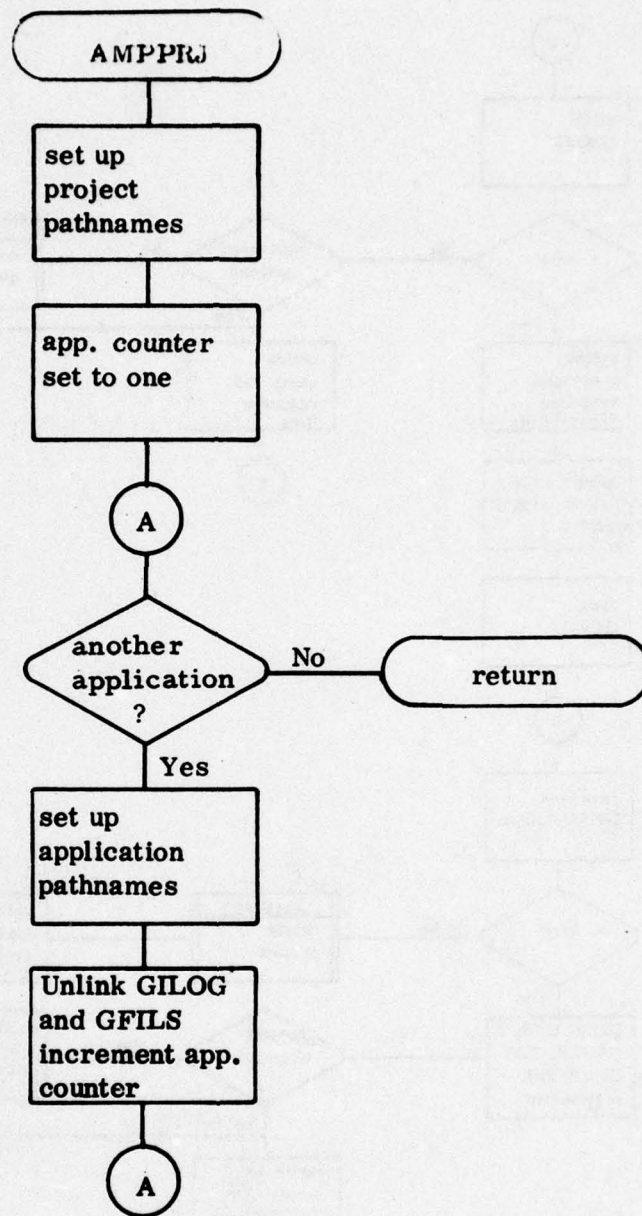


Figure 11. AARMVSN I Process Data Flow (Sheet 5 of 6)

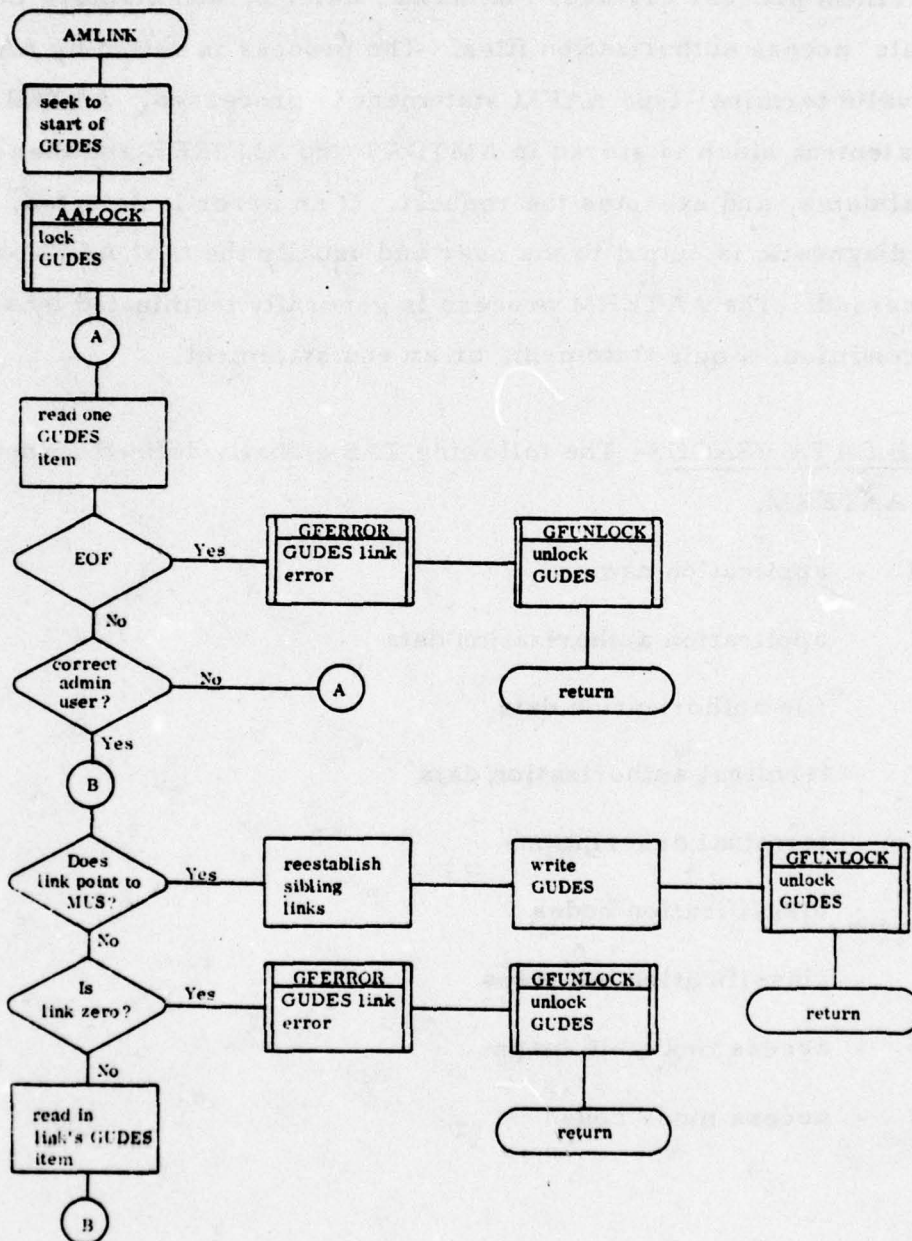


Figure 11. AARMVSNI Process Data Flow (Sheet 6 of 6)

AATERM (Terminal AA File Maintenance)

The AATERM process creates, maintains, deletes, and displays the TAS terminals' access authorization files. The process is called by AAFM whenever a valid terminal-type AAFM statement is processed. AATERM reads the statement which is stored in AMTDAT and AMTREE and then analyzes, validates, and executes the request. If an error is detected, an informative diagnostic is output to the user and usually the next AAFM statement is processed. The AATERM process is generally terminated by an end-of-file condition, a quit statement, or an end statement.

GLOBAL DATA USAGE - The following TAS globally defined structures are used by AATERM.

GAPNM	- application names
GAPPL	- application authorization data
GFILS	- file authorization data
GTAUT	- terminal authorization data
GTERM	- terminal descriptions
GCLAS	- classification codes
GMAP	- classification bit-maps
GAMAP	- access mode bit maps
GACCM	- access mode codes

LOCAL DATA USAGE - The following list describes the significant local data used by AATERM.

- AMTREE - AATERM uses the parse tree AMTREE for statement processing. In this integer array predictably placed nodes correspond to the pertinent elements in a statement.
- AMTDAT - The AMTDAT character array holds data values for a specific statement. AMTDAT is indexed by pointers stored in the parse tree (AMTREE).
- AMTTIND - The terminal index for GTERM is also called the terminal's TAS Station Number.
- AMTTYAR - The terminal identifier is a single ASCII character.
- AMAPIND - AMAPIND is the index of the application which is currently being created or changed. AMAPIND is used for the index into the GAPPL file.
- AMFLIND - AMFLIND is the index of the file which is currently being created, changed, or deleted. AMFLIND is an index into GFILS.
- AMBRFL - The branch flag AMBRFL is true whenever the statement currently stored in AMTREE and AMTDAT has to be further processed. The necessity of the flag is due to AATERM's hierarchically structured processing of applications and files.

- AMBASIC - A flag which, if set, indicates that the basic terminal information is to be output. (Syntactically, this implies that neither a terminal identifier nor a TERMLIST is specified in the list-terminal statement.)
- AMALL - A flag which, if set, indicates that all access authorization data are to be output. (Syntactically, the list-terminal statement has a terminal identified.)

GENERAL PROCESS FLOW

When AAFM processes a terminal-type statement, it performs basic validation which, if passed, results in the calling of AATERM to process the statement's request. AAFM passes two pipe file descriptors as a mechanism for AATERM to communicate with the AALOCK daemon. Having stored the file descriptors, AATERM reads GTERM, AMTREE, and AMTDAT. The latter two files (built by AAFM) contain the parse tree data and the character data pertinent to the terminal-type statement. The parse tree AMTREE is used to determine the statement type and a switch is made on the type to process the statement further.

The create-terminal statement is processed by the AMCRT function. The entire create-terminal command block is basically handled by this function. The delete-terminal statement is validated to determine whether or not the specified terminal exists and if so, whether it is in use (a user is logged onto the terminal). If the terminal does exist and it is not in use, the terminal is removed from the TAS environment by function AMRTERM. The change-terminal statement which initiates a change-terminal command block is processed by the AMCHT function.

The execute-terminal statement causes the UNIX external variable FIN to be set to the text file which is specified in the execute statement. An execute statement cannot be embedded within the specified file.

The processing of the list-terminal statement is essentially done in this main function. The list-terminal statement has four basic ways to display terminal data:

1. A list-terminal statement without any options results in the display of all TAS terminals' basic information. This information includes the TAS station number, terminal (character) identifier and owner.
2. A list-terminal statement with just a TERMLIST option causes the specified data (as given in the TERMLIST) to be output for each TAS terminal.
3. A list-terminal statement with just a terminal specified results in the displaying of all data stored for the given terminal.
4. A list-terminal statement with both a terminal identifier and a TERMLIST given causes the desired data for the terminal to be output.

The following outline describes the processing of the four variations of the list-terminal statement.

1. For the list-terminal statement without any input options, the flag AMBASIC is set true. For each terminal defined in GTERM, the terminal's AA files GTAUT and GAPPL are read and function AMLSTT is called to display the terminal's basic information.
2. When a terminal is not specified but a TERMLIST is given, the GTERM file is read. For each defined TAS terminal, its GTAUT and GAPPL files are read and the function AMLSTT is called to output the terminal's access authorization data as requested in the TERMLIST.
3. When a terminal is specified in a list-terminal statement, a match on the input terminal identifier is attempted in GTERM. If the terminal does exist, its GTAUT and GAPPL files are read. Since a TERMLIST is not given by the requester, the flag AMALL is set

true, indicating that all information concerning the terminal is to be displayed by function AMLSTT. If the terminal does not exist, a diagnostic is output.

4. If both a terminal and a TERMLIST are specified, a match on the input terminal identifier is attempted in GTERM. If the terminal does not exist, a diagnostic is output to the TASMATER. If, on the other hand, the terminal is defined in GTERM, its GTERM and GAPPL files are read. The AMLSTT function is called to output the terminal's access authorization data as requested in TERMLIST.

Having displayed the requested terminal access authorization data, the flags AMBASIC and AMALL are set false.

After processing the terminal-type statement, a check is made for the possibility of one of two conditions: an end-of-file condition or input-from-a-terminal condition. If either condition is true, the AATERM process exits with a normal status. If, however, the input is from a text file, AATERM calls the LXINIT function to parse the next AAFM statement in the text file and to build the parse tree and parse tree data files. Upon return to the main function from LXINIT, a switch is made on the statement-type (CREATE TERM, DELETE TERM, LIST TERM, CHANGE TERM and QUIT) and the procedure described above is repeated. The only new additional statement is the QUIT command which causes an exit of the AATERM process with a quit status. (The AAFM process will terminate as a result of such a return status.) It might be noted at this time that the execute-terminal statement is no longer a valid, recognized statement by AATERM. It must always be the first statement processed by AATERM in order to be valid. (See figure 12 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

AMCRT (Create Terminal A Files) - The AMCRT function processes the create-terminal command block which is initiated by the create-terminal statement and terminated by the end statement. AMCRT ensures that

terminal identifiers are unique in the TAS environment, as well as controls which statement types are allowed in the create-terminal command-block. The input terminal identifier is validated to be the character string 'TTY' followed by one alphanumeric character. If the terminal identifier is not valid, a diagnostic is output and the function returns to the calling function. If the terminal identifier is syntactically valid, AMCRT locks (via the AALOCK daemon) GTERM. GTERM is read and a sequential search is made to determine that the single character identifier (AMTTYAR) is not currently defined in GTERM. If it is already present, an error message is output and AMCRT returns; otherwise, the new terminal identifier is written into the first available (empty) slot in GTERM. This index is stored in AMTTIND, and then GTERM is unlocked. The terminal directory is made and files GTAUT and GAPPL are created for the terminal.

AMCRT now calls the LXINIT function which parses the next incoming statement and builds the AMTREE and AMTDAT files. Below is outlined the functions called depending on the statement types.

1. Add application statement - Call AMAAPP to add either an interactive or a batch application to GAPPL.
2. End statement - Call AMETERM to end the definition of the terminal and to write out the terminal's AA files.
3. Add compartment statement - Call AMCOM to add new compartment names.
4. Location, clearance, telephone or organization statement - Call AMBT to store general terminal data.
5. Other statements - Output an error message.

If one of the called functions had set the reprocess-statement flag AMBRFL to true before returning to AMCRT, the process control jumps to the statement type switch (discussed above) to reprocess the statement stored in the parse tree files. Otherwise, the LXINIT function is called to syntactically validate and parse the next incoming statement.

AMCHT (Change Terminal AA Files) - The AMCHT function processes the change-terminal statement and if valid, processes the remaining statements in the change-terminal command block. The function AMRDT validates the input terminal identifier. If it is not a valid one, an error message is output, GTERM is unlocked, and the function returns to the calling function. If the terminal identifier is valid, AMCHT then calls LXINIT to syntactically validate the next incoming statement and to build the parse tree files. Upon return, AMCHT does a switch on the statement type. The switch for the AMCHT function is the same as the switch for AMCRT with the addition of the following three statements.

1. Change application statement - Call AMUAPP to update batch application access authorization data.
2. Delete application statement - If the application is defined for the terminal, call AMRAPP to remove the application; otherwise, output a diagnostic to the user.
3. Delete compartment statement - Call AMCOM to delete compartment names.

Upon return to AMCHT, a check is made on the reprocess-statement flag AMBRFL. If the flag is set, AMCHT branches back to the statement type switch which was discussed above; otherwise, the LXINIT function is called to syntactically validate and parse the next incoming statement.

The end-statement and the end-of-file condition both result in the calling of AMETERM to write the terminal files and a return to the calling function.

AMRDT (Validate Input Terminal Identifier) – AMRDT is called to validate the input terminal identifier for the CHANGE TERM, LIST TERM and DELETE TERM statements. If the GTERM file is not locked, GTERM is locked by the AALOCK daemon and is then read. The terminal identifier is in the form of either a TAS Station Number or the character string TTY followed by an alphanumeric character. If the identifier is valid, a scan of GTERM determines whether the identifier is valid for a currently defined TAS terminal. If the terminal identifier is not valid, an error message is output and a negative value is returned to the calling function. If the statement type is either CHANGE TERM or DELETE TERM, further validations must be made. The terminal's logged on status as stored in GTERM.GTLOG must be logged off unless the specified terminal is the one the TASTMASTER is presently using. If the terminal is being used by someone other than the TASTMASTER, a diagnostic to that effect is output and a negative value is returned. Otherwise, the terminal status is changed to reflect an update status and the GTERM file is written and then unlocked. If the statement is CHANGE TERM, the files GTAUT and GAPPL are read. For all three statement types, a positive value is returned to the calling function.

AMETERM (End Terminal AA Files) – The AMETERM function first writes GAPPL and GTAUT, then calls the AALOCK daemon to lock GTERM. The terminal's GTERM.GTLOG is set to a status of logged off and the GTERM file is written and unlocked. The AMETERM function returns to the calling function.

AMBT (Basic Terminal Information) – The basic terminal information statements are processed by the AMBT function. This function is called by either AMCRT (create terminal function) or AMCHT (change terminal function). The four statements listed below are processed by AMBT and their information is stored appropriately in GTAUT:

1. Location statement - The location of the terminal is input by means of a character-constant whose length cannot exceed one hundred characters. The data is stored in GTAUT.GTLOC.
2. Telephone statement - The telephone number is input as a character-constant with a maximum length of twenty characters. The number is stored in the character array GTAUT.GTTEL.
3. Organization statement - The organization to which the terminal is assigned is input as a character-constant with a maximum length of ten characters. This data is stored in GTAUT.GTOWN.
4. Clearance statement - The single-character input clearance code is checked by the LEX/parse process to determine whether or not it is a valid security clearance level. The bit map for the specified clearance code is retrieved from GMAP which is parallel in construction to GCLAS, thus allowing the capability of cross-referencing. The bit map is stored in GTAUT.GTCLR.

AMCOM (Process Compartment Names) - The function AMCOM processes the add and delete compartment statements. Each terminal may have a maximum of fifty compartments assigned to it. The AMCOM fully processes each valid compartment name in the list of input names, and outputs an error message for each invalid name. A compartment name has a maximum length of six characters. If the MP is adding a new compartment name, the name must differ from existing compartment names. In contrast, when the MP deletes a compartment name, the name must be an existing compartment name. The list of compartment names is stored in a two-dimensional array, which is a variable length list of six character compartment names. A count is kept of the number of currently defined compartment names.

AMRTERM (Remove Terminal AA Files) - The removal of a terminal from the TAS environment is accomplished by AMRTERM. AALOCK is called to lock GTERM. The terminal's status GTERM.GTLOG is set to a status of CREATE/DELETE, if not currently set to that value. To remove the terminal's application access files, the AMRTERM function prepares the

pathnames and unlinks file GFILS for each application defined for TAS. Next the GAPPL and GTAUT files and the tty directory are unlinked. The terminal's GTERM entry is cleared and written out. Finally GTERM is unlocked and the AMRTERM function returns to the calling function.

AMAAPP (Add Application) – The function AMAAPP adds an application to the terminal's AA files. The application name in consideration cannot be an application already existing for the terminal; if it is, an error message is output and AMAAPP returns to the calling function. If the application is unique, the application index AMAPIND is set to the first available slot in GAPPL.

If the application is interactive, just the application index is stored in GAPPL and the function AMAAPP returns to the calling process.

If the application is batch, the AMAAPP function creates GFILS and calls LXINIT which parses the next input statement. When LXINIT returns, AMAAPP switches on the statement token type. If the statement is an add-file statement, it calls the AMAFIL function which processes the statement. Otherwise, it calls the AMEAPP function (which validates the condition of the application definition), sets AMBRFL to true, and returns to the calling function. The AMAAPP function continually processes add-file statements until another statement type is input.

AMUAPP (Update Application) – The function AMUAPP is called whenever an application is being either changed or deleted. The AMUAPP function picks up the application name and determines whether it is currently defined in GAPPL for the terminal. If not, an error message is printed out and AMUAPP returns to AMCHT.

If the statement type is delete, the function AMRAPP is called to delete the terminal.

If the application is batch and the statement type is CHANGE APP, the LXINIT function is called to parse the next incoming statement, and upon returning, a switch is made on the statement type. The add-file statement causes the AMAFIL function to be called; likewise, a change- or delete-file statement causes the calling of the AMUFIL function. All other statements cause a call to AMEAPP (which validates the existence of the application's necessary data), the reprocess statement flag AMBRFL to be set, and a return to the calling function.

The AMUAPP function processes add-, change- or delete-file statements until another statement type is detected.

AMEAPP (Validate Application Data) – A batch application definition must be validated to ensure the existence of necessary information. For each batch application at least one file must be defined. If it is not, the GFILS file is unlinked, its entry in GAPPL deleted, and the MP is informed. However, if all is in order, the application file GFILS is written out and the function returns to its calling function.

AMRAPP (Remove Terminal Application Access Authorization) – An application is removed from a terminal's GAPPL file by AMRAPP. The application file GFILS is unlinked whether or not it actually exists. The GAPPL entry is cleared and the application count is decremented to the index of the last currently used entry in GAPPL. The function AMRAPP returns to the calling function having removed the application from the terminal's AA files.

AMAFIL (Add File) — A file is defined with respect to a batch application. An add-file statement contains a filename which is input in the form of a character-constant and is validated to be a unique name within an application definition. The maximum length of the name is ten characters. The next step is to determine the input access mode bit map via GAMAP. The AMAFIL function stores the filename and access mode in GFILS and returns to the calling function.

AMUFIL (Update File) — The changing or deleting of data concerning an existing file is done in the AMUFIL function. The filename is validated to be an existing filename in GFILS for a given application.

If the statement type is DELETE FILE, the file data entry is removed from the GFILS file. AMUFIL returns to the calling function.

When the statement type is CHANGE FILE, the access mode bit map is determined and AMUFIL stores the file data in GFILS and returns to the calling function.

AMLSTT (List Terminal's AA Files) — The AMLSTT function clears the screen and then displays the terminal's basic information: the TAS station number, UNIX terminal identifier, and owner. If the AMBASIC flag is set, AMLSTT returns to the calling function. If AMALL is true, the following terminal data is also output in the listed order:

1. Clearance
2. Location of the terminal
3. Telephone number
4. Number of compartments
5. Compartment names
6. Applications
7. Batch Files and Access Modes

Having output the contents of the terminal's AA files, the AMLSTT function returns.

If the MP had specified output via a TERMLIST, the tokens stored in the list are processed and the data are output in the order in which the TERMLIST was input and built. A switch is made on each token type and the appropriate information is output. The telephone number, location of the terminal, organization to which the terminal belongs, and compartment names (which are stored as characters arrays in GTAUT) are simply transferred into a buffer and output. The terminal's clearance bit map, stored in GTAUT. GTCLR, has to be matched to a bit map in GMAP. The index for GMAP is used to index GCLAS which contains the descriptive mnemonics for the classification bit map. This information is output on the screen.

When the MP specified APP in the TERMLIST, just the host names to which the terminal has access are output. If the token FILE is in the TERMLIST, the application names are output and if the application is a batch host, the host's files and their access modes are also output.

After processing each token in the TERMLIST for any given terminal, the function AMLSTT returns to the calling function.

LXINIT (Lex/Parse Initialization) — LXINIT is merely an initialization function for the lex/parse portion of this process. It initializes pertinent data including a flag which is returned to the calling function to indicate end-of-file (EOF), syntax error, or statement accepted. LXINIT calls the parsing function (YYPARSE) and returns with the flag value.

YYPARSE (Parser) — YYPARSE is a parser which is produced by the Yet Another Compiler-Compiler (YACC) program running under UNIX. YACC produces the parser as well as a set of tables which the parser uses to organize the tokens passed to it by the lexical analyzer (YYLEX). These

tables reflect the grammar of the various AATERM statements. YYPARSE calls YYLEX, which returns a value called a token type. If the token type is invalid according to the input statement syntax rules, an error message is output and YYPARSE calls YYLEX continuously until an end-of-input token type is returned. YYPARSE then returns with the return flag set to reflect a syntax error. If the token type is valid according to the input rules, the action specified in the YACC run is performed. The actions that can be invoked are: no action, build a parse tree node with the specified number of legs, store the final token type and write out the parse tree and parse tree data, start a list, add an element to the current list, and store the list in the parse tree. After the action is performed, YYPARSE calls YYLEX for the next token type. When the end-of-input token type is received and its action performed, YYPARSE returns.

YYLEX (Lexical Analyzer) - YYLEX is the function which actually inputs and processes each AATERM statement. YYLEX reads and saves each input character until a delimiter is encountered.

The delimiters for AATERM are: comma, semi-colon, newline (carriage return), single quote mark, and space. If an EOF is encountered, the return flag is set to reflect EOF and an end-of-input token type is returned. If a delimiter is encountered by itself, it is either passed to the parser (comma, or semicolon), it is ignored, or it causes YYLEX to change its input mode; e. g. , single quote mark indicates a character-constant input and causes YYLEX to read characters until another single quote mark is input. When a delimiter terminates a string of characters, YYLEX must determine what the characters represent. If the characters form an AATERM keyword, YYLEX returns with that keyword's token type. Otherwise, YYLEX determines the type of constant which is formed by the characters (name, integer, or character-constant), stores the value of the constant in the parse tree data (AMTDAT), stores a terminal node in the parse tree

(AMTREE), and returns with the constant's token type. If YYLEX encounters an error (e.g., too many characters input), it returns an invalid token type to the parser and thereby generates a syntax error message.

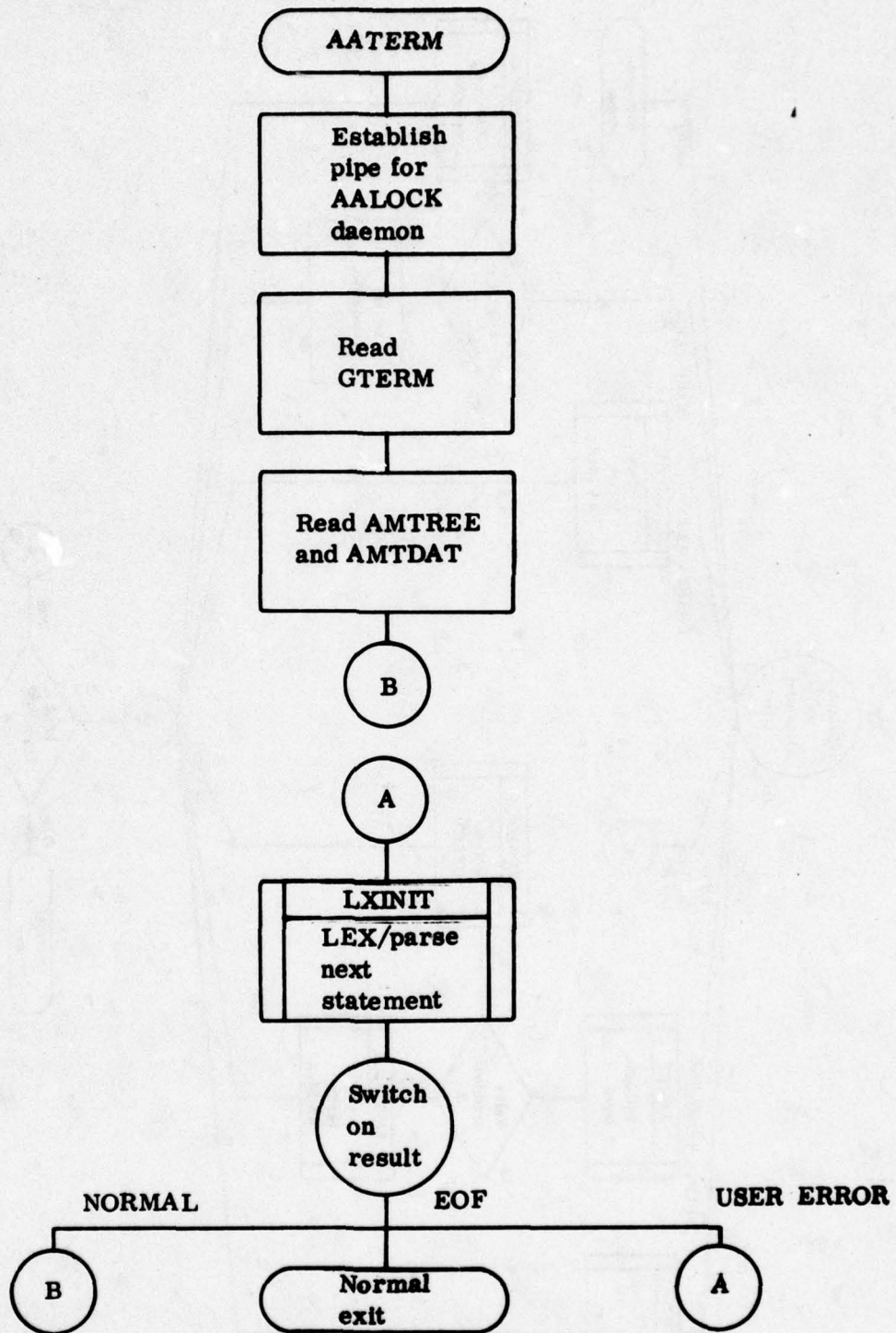


Figure 12. AATERM Process Data Flow (Sheet 1 of 19)

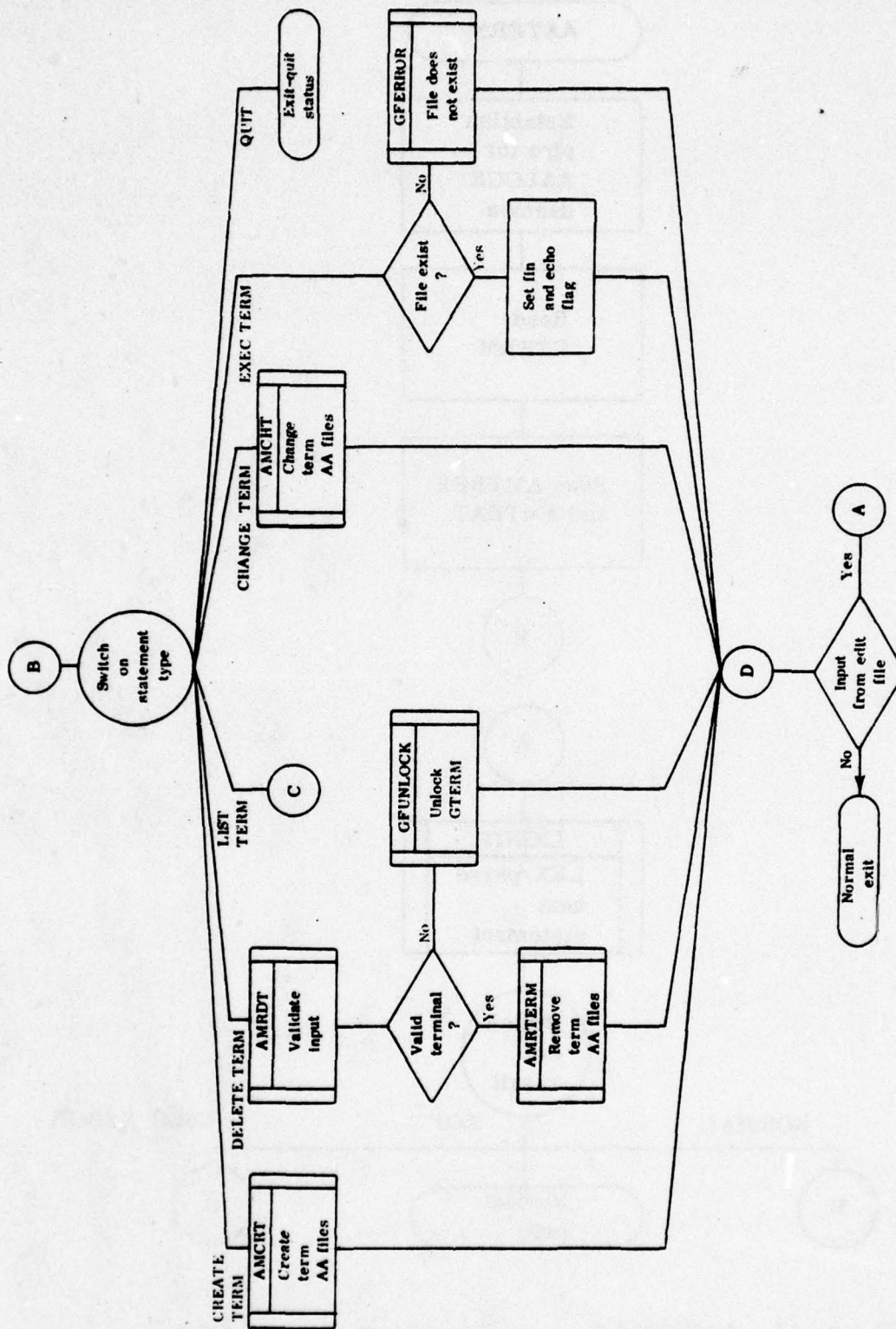


Figure 12. AATERM Process Data Flow (Sheet 2 of 19)

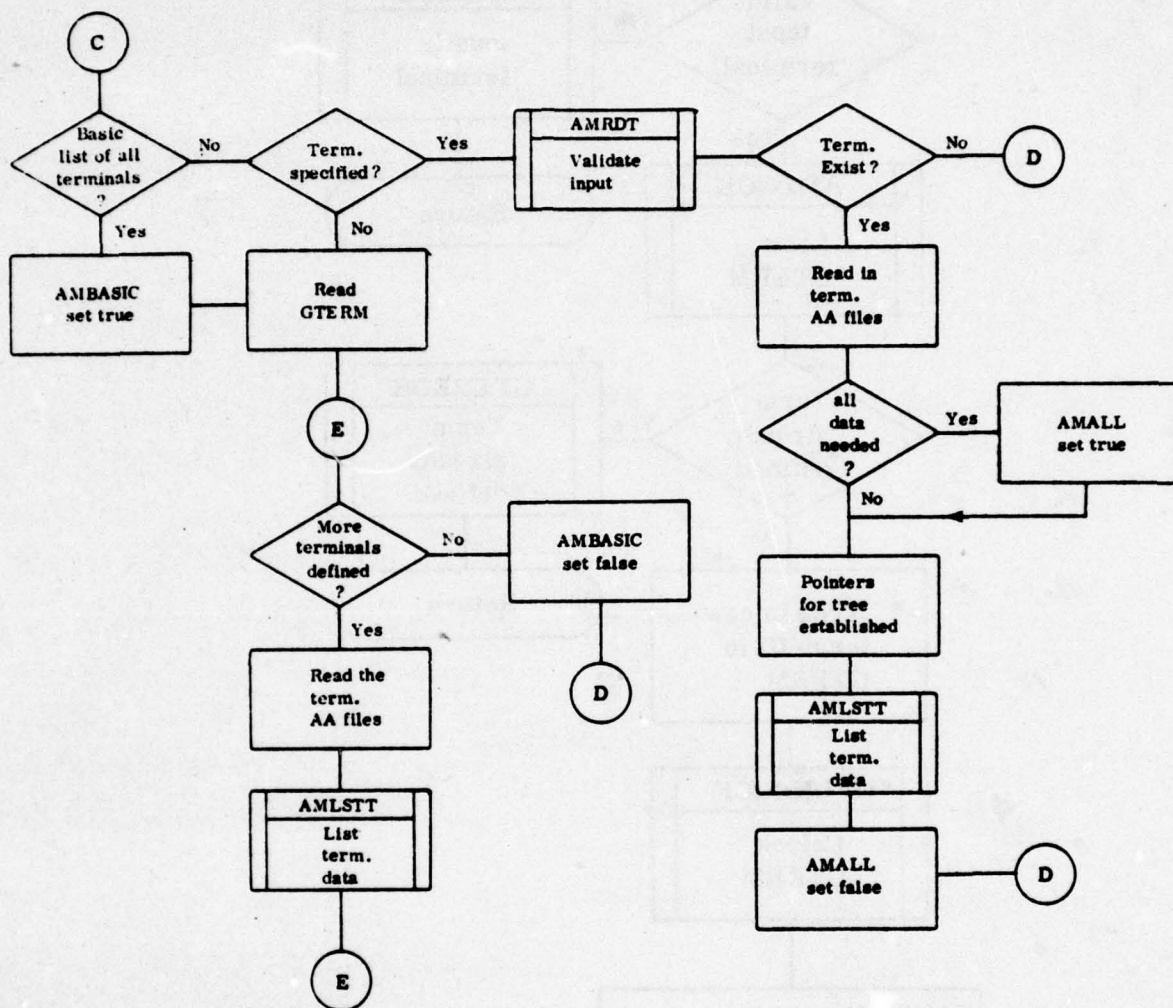


Figure 12. AATERM Process Data Flow (Sheet 3 of 19)

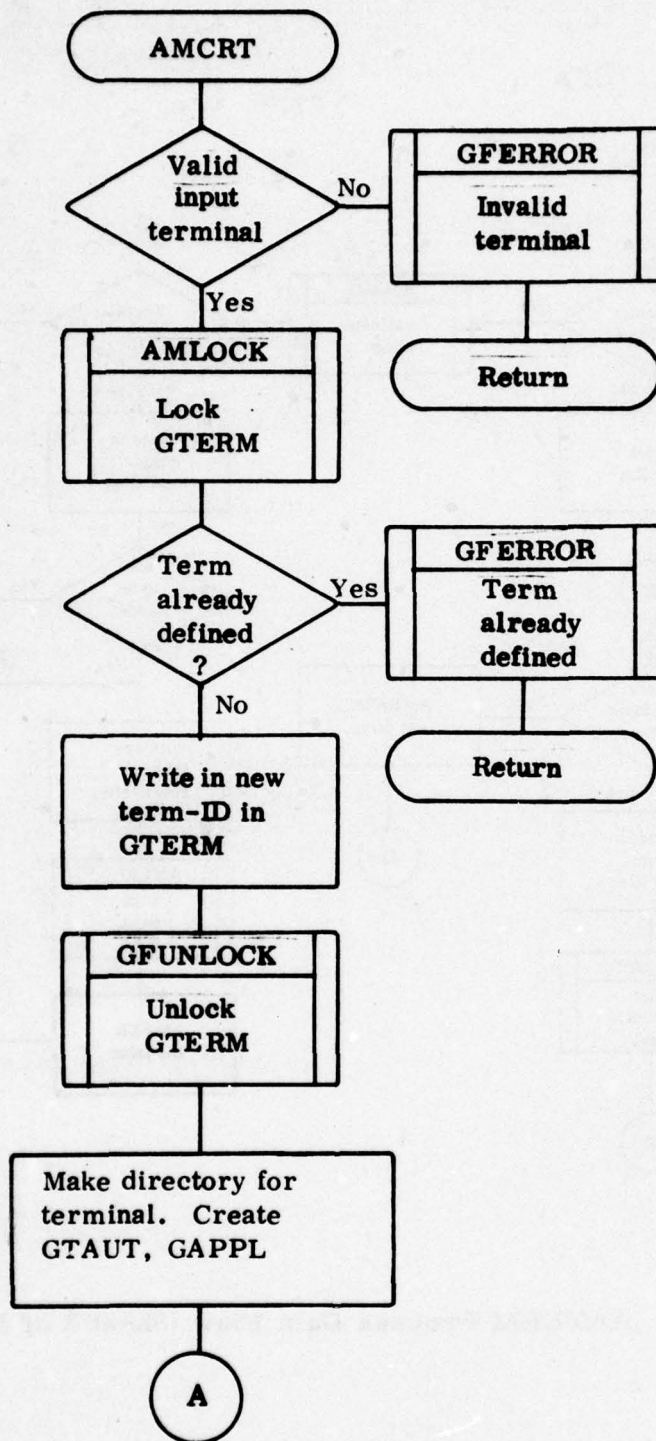


Figure 12. AATERM Process Data Flow (Sheet 4 of 19)

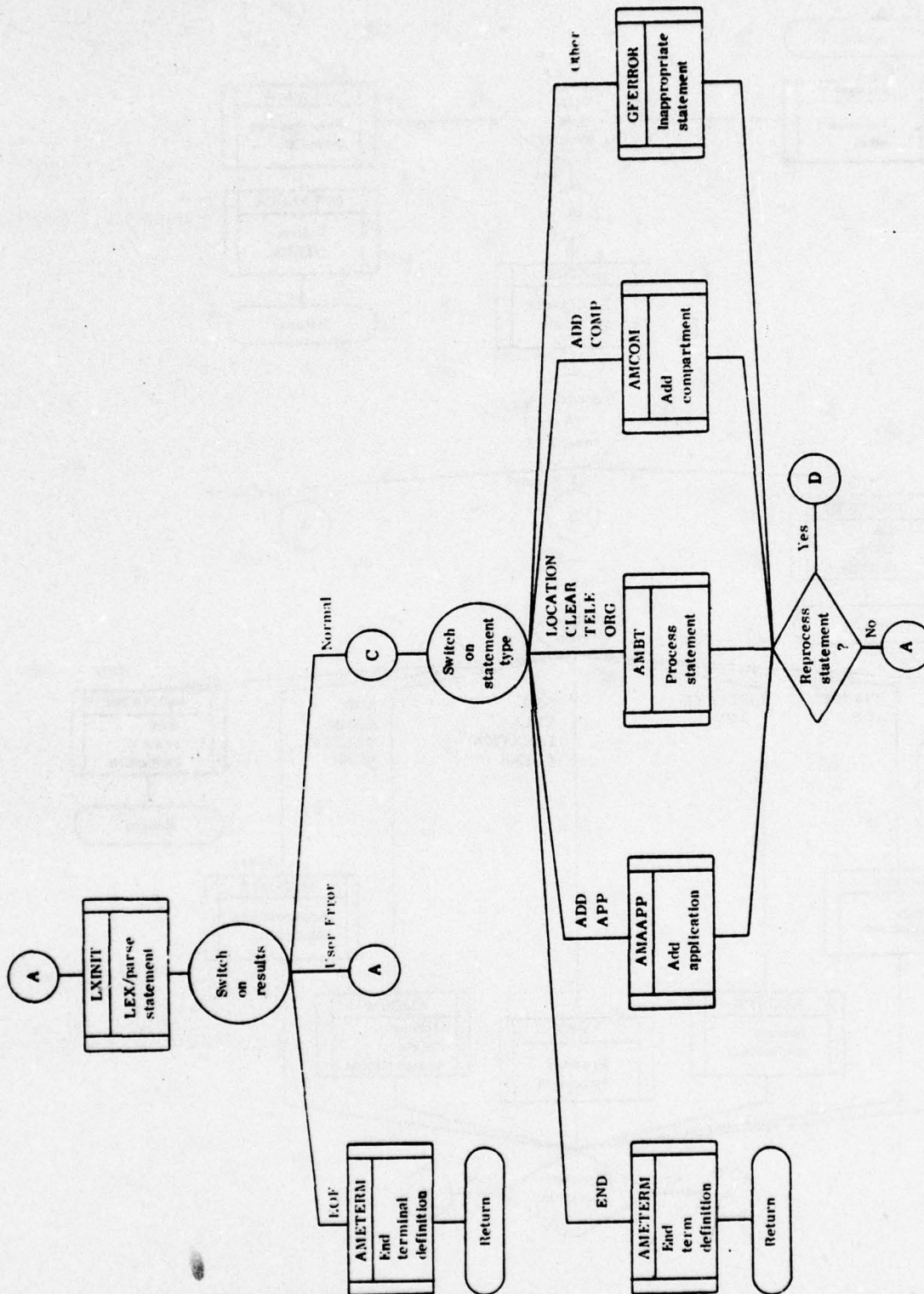


Figure 12. AATERM Process Data Flow (Sheet 5 of 19)

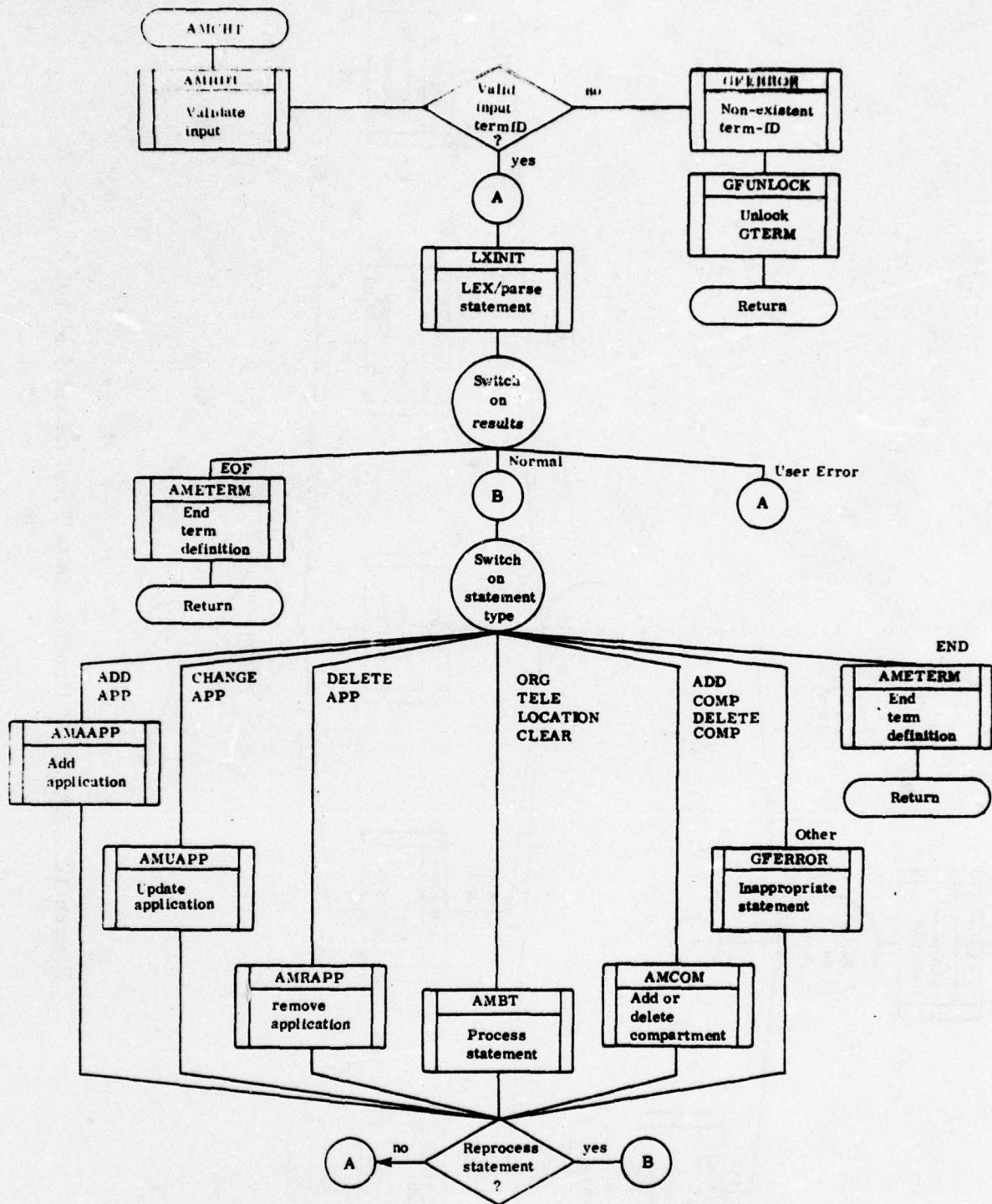


Figure 12. AATERM Process Data Flow (Sheet 6 of 19)

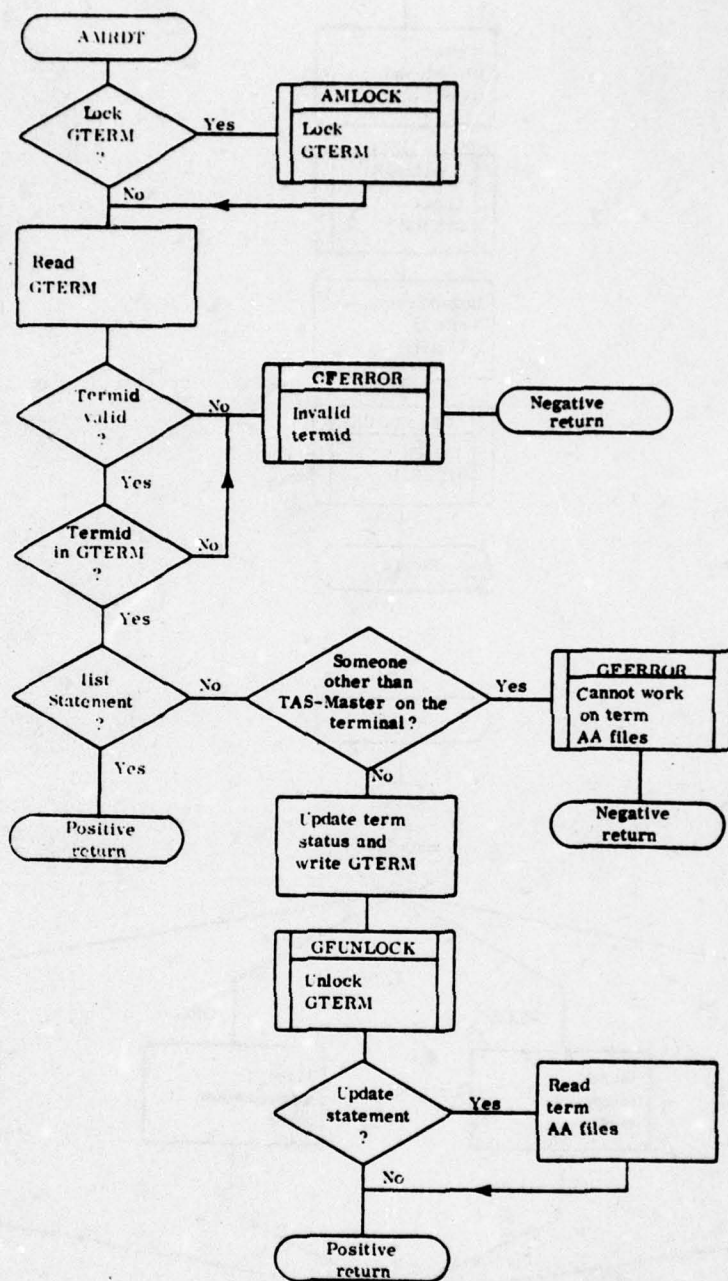


Figure 12. AATERM Process Data Flow (Sheet 7 of 19)

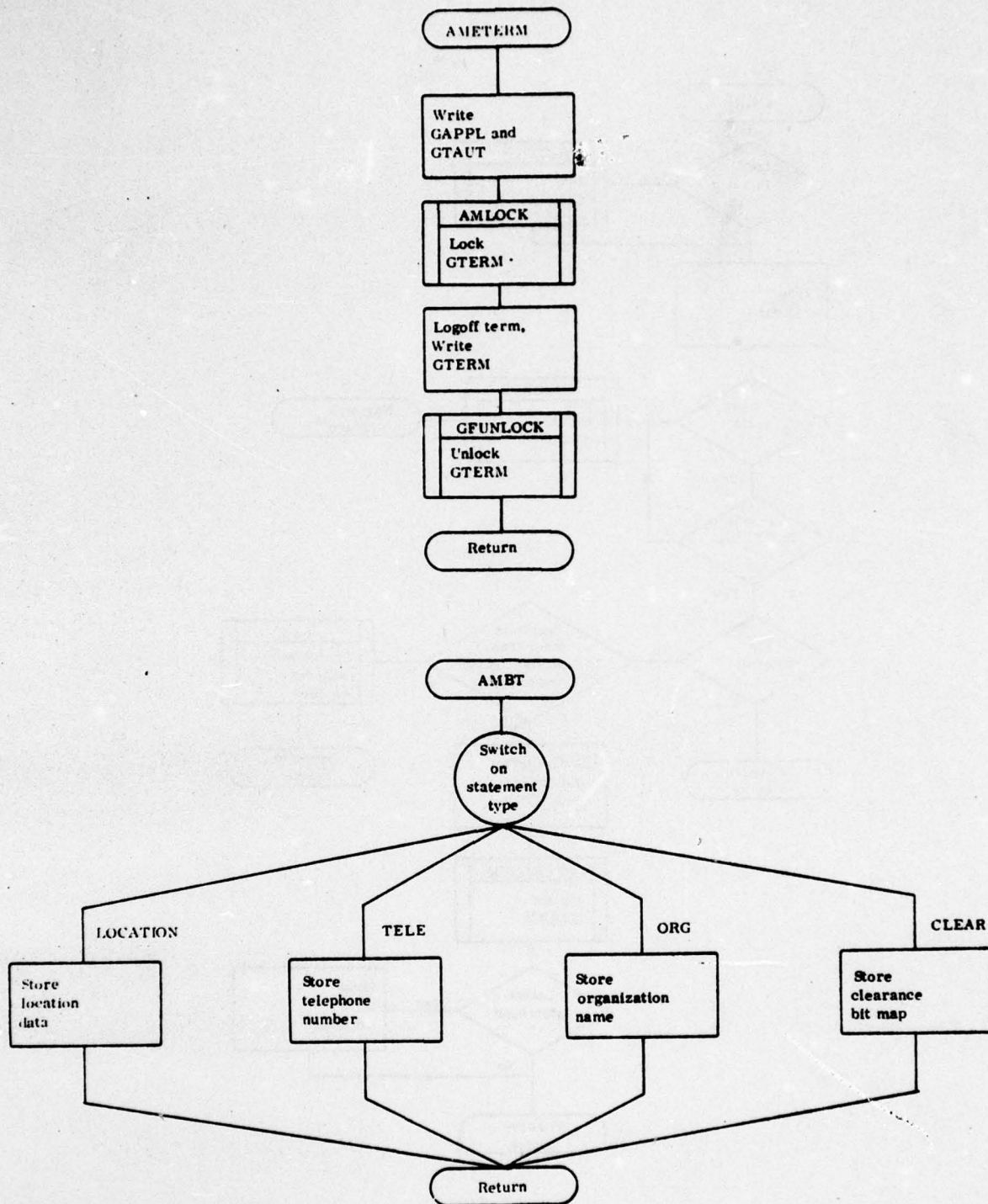


Figure 12. AATERM Process Data Flow (Sheet 8 of 19)

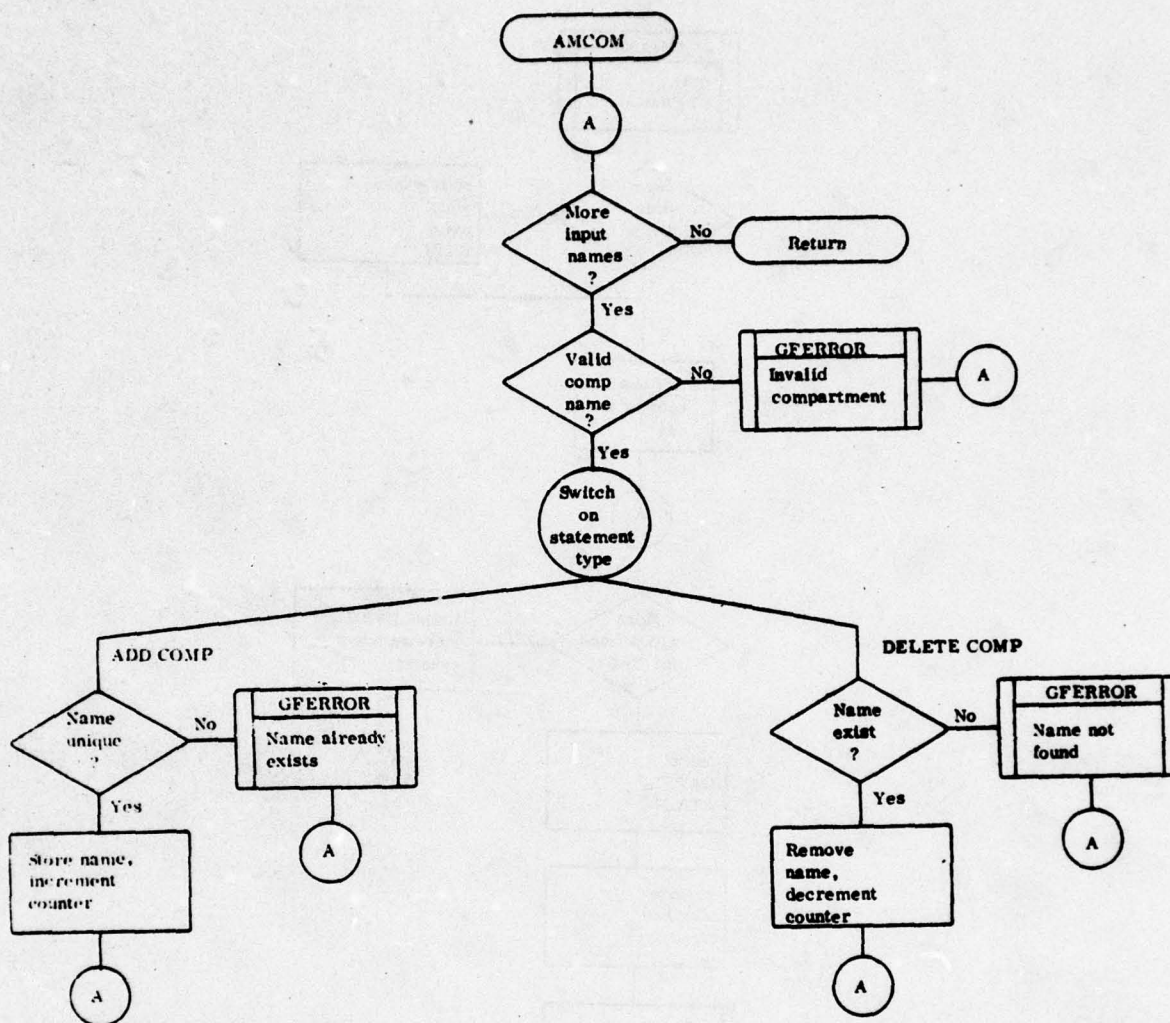


Figure 12. AATERM Process Data Flow (Sheet 9 of 19)

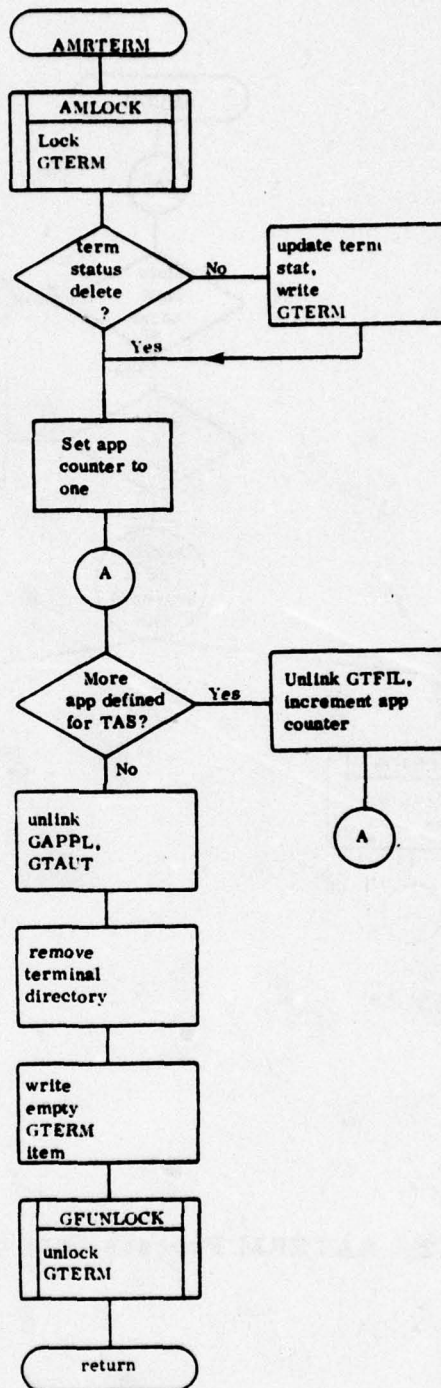


Figure 12. AATERM Process Data Flow (Sheet 10 of 19)

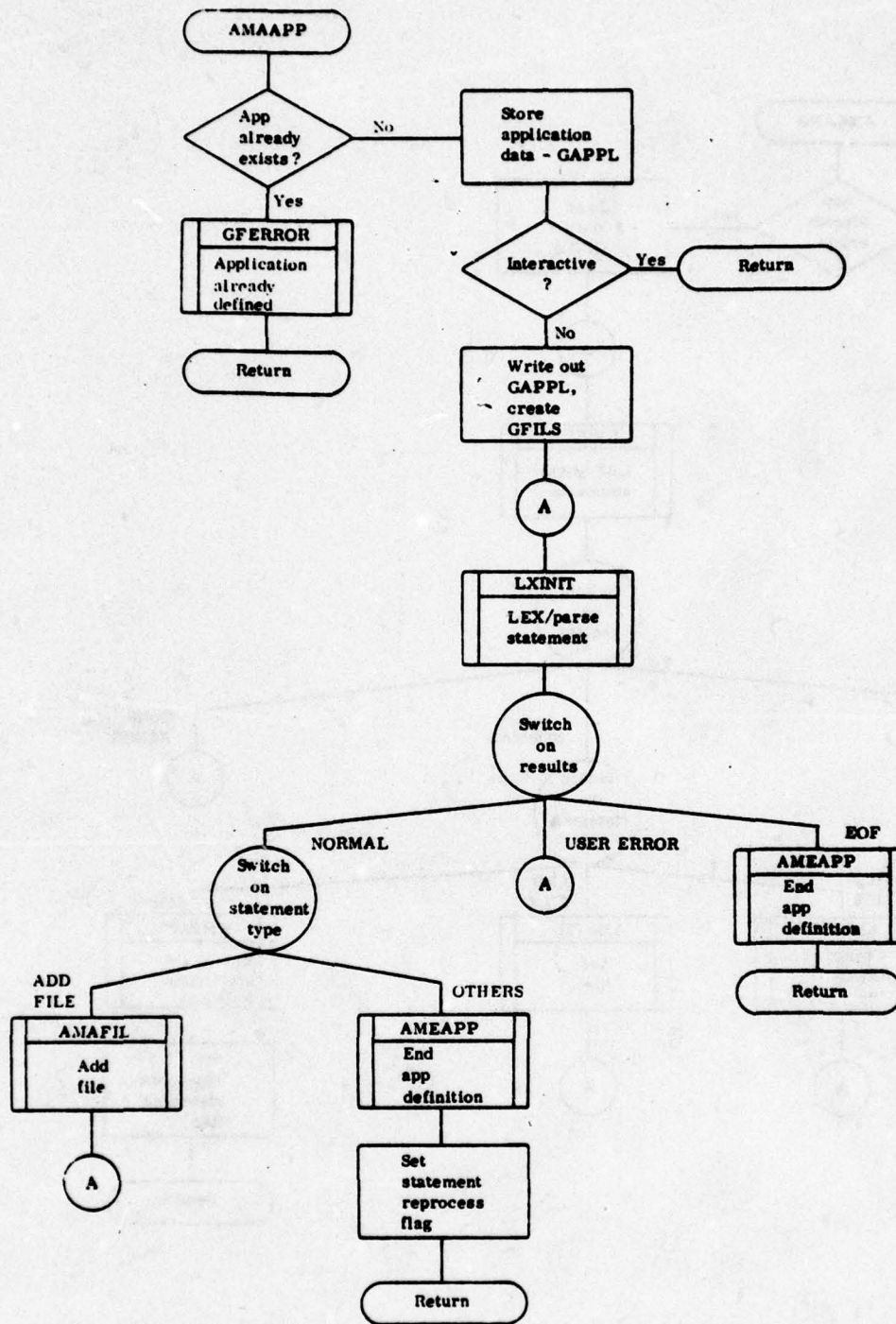


Figure 12. AATERM Process Data Flow (Sheet 11 of 19)

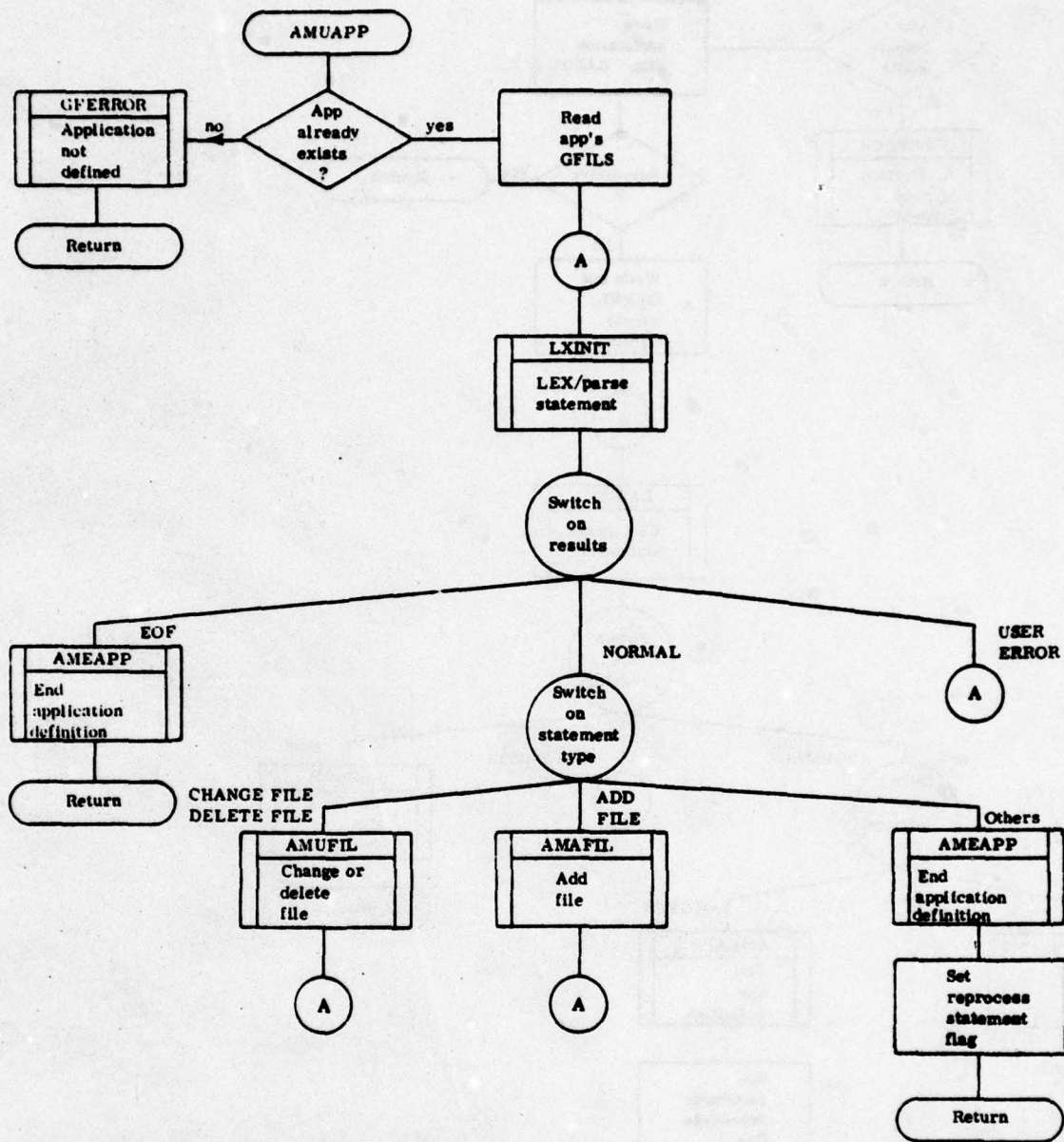


Figure 12. AATERM Process Data Flow (Sheet 12 of 19)

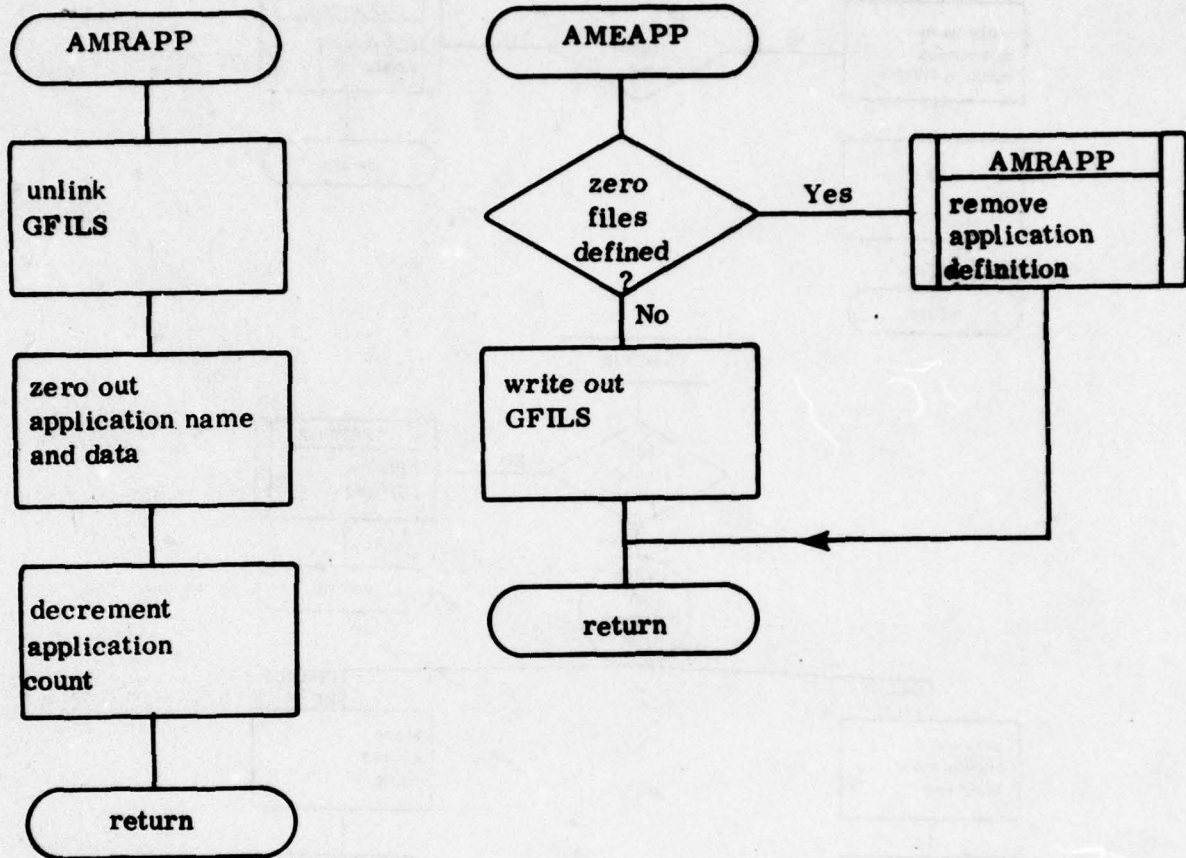


Figure 12. AATERM Process Data Flow (Sheet 13 of 19)

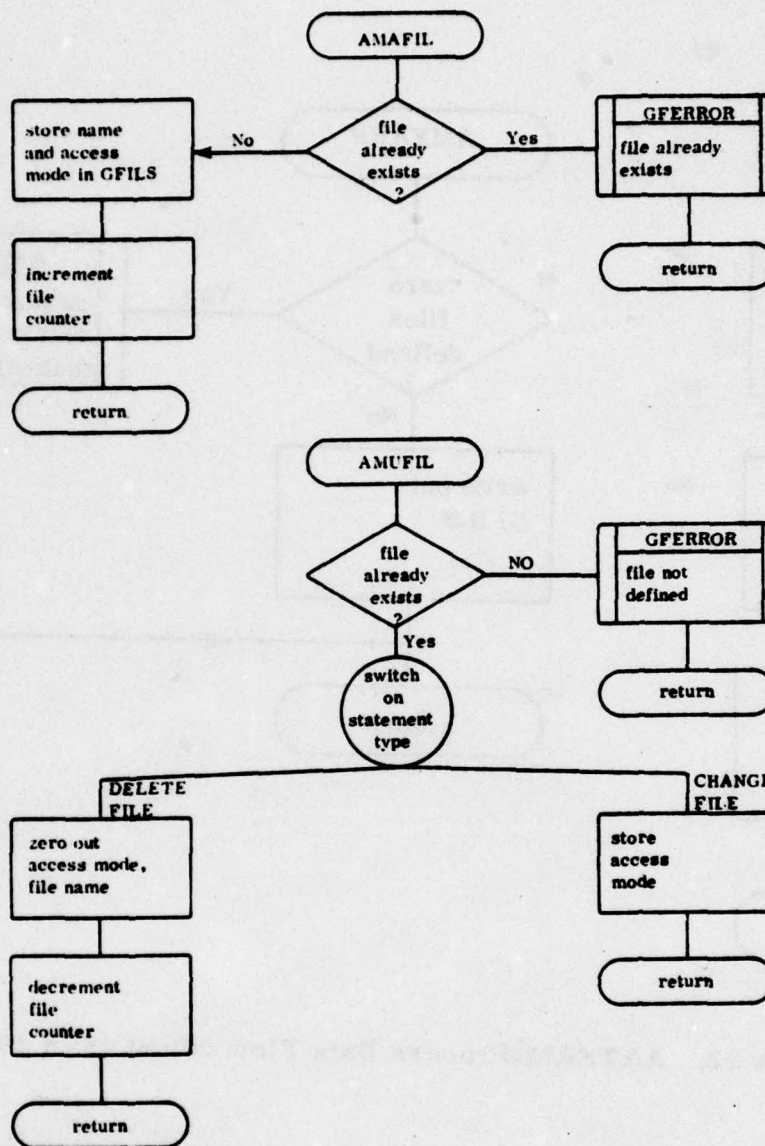


Figure 12. AATERM Process Data Flow (Sheet 14 of 19)

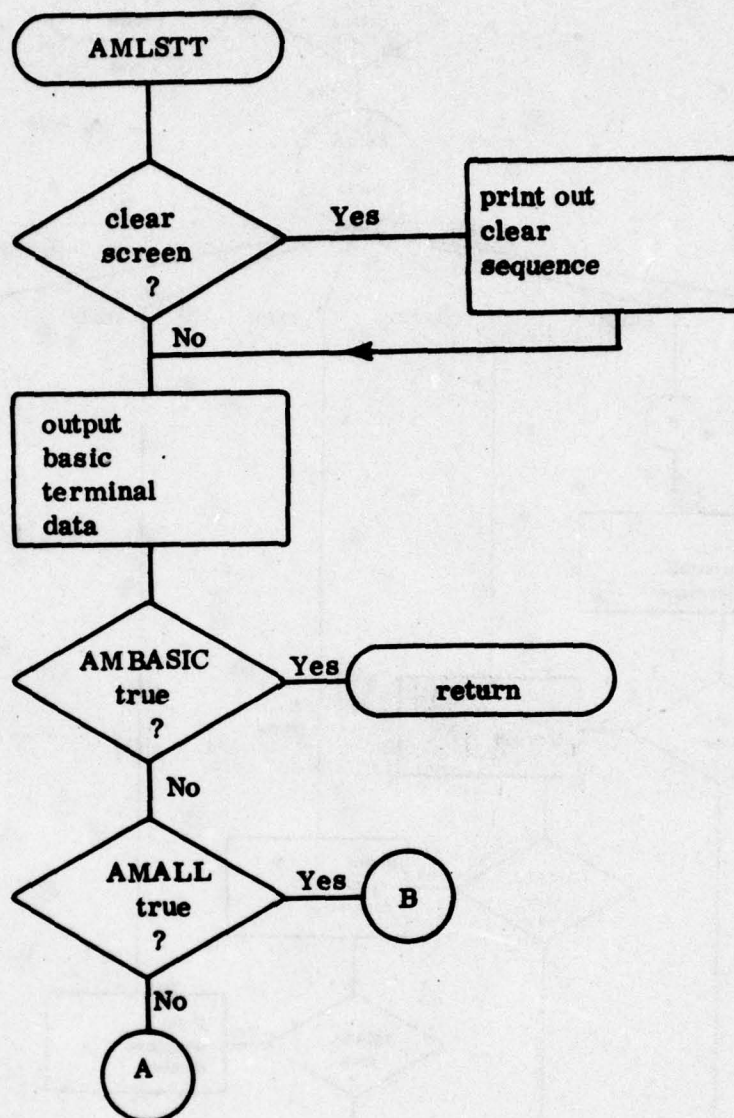


Figure 12. AATERM Process Data Flow (Sheet 15 of 19)

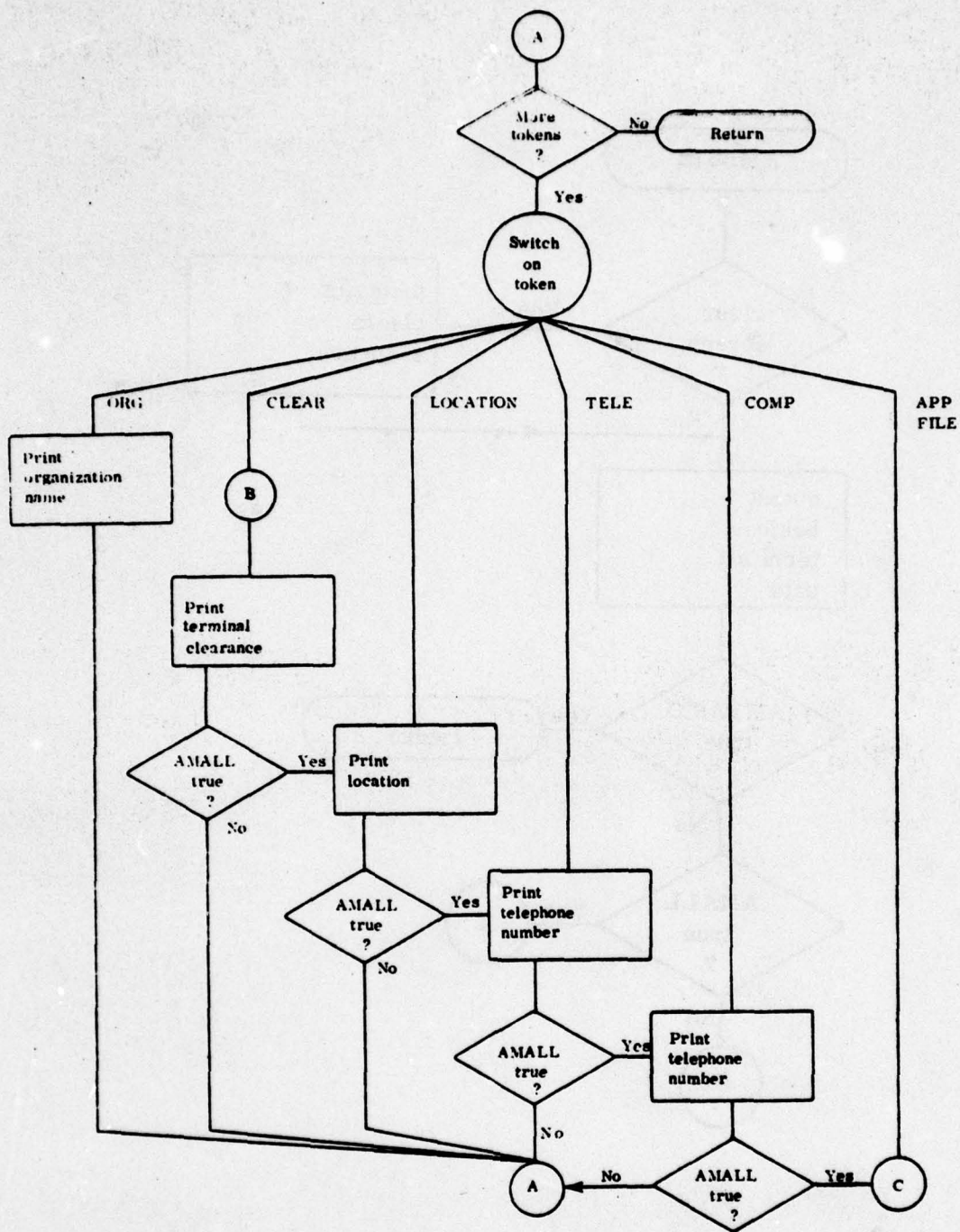


Figure 12. AATERM Process Data Flow (Sheet 16 of 19)

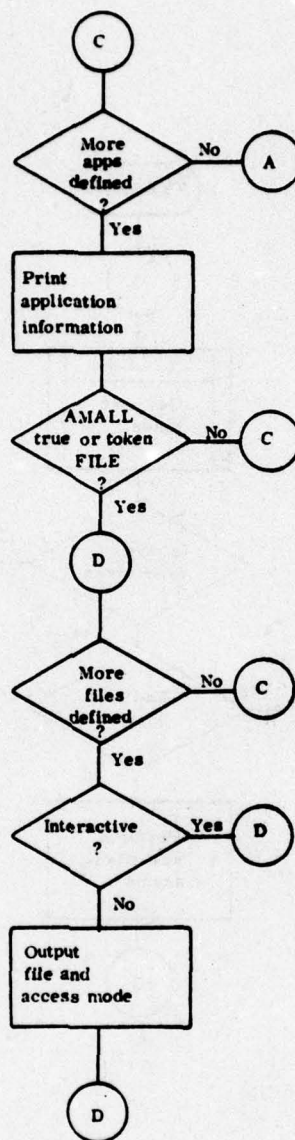


Figure 12. AATERM Process Data Flow (Sheet 17 of 19)

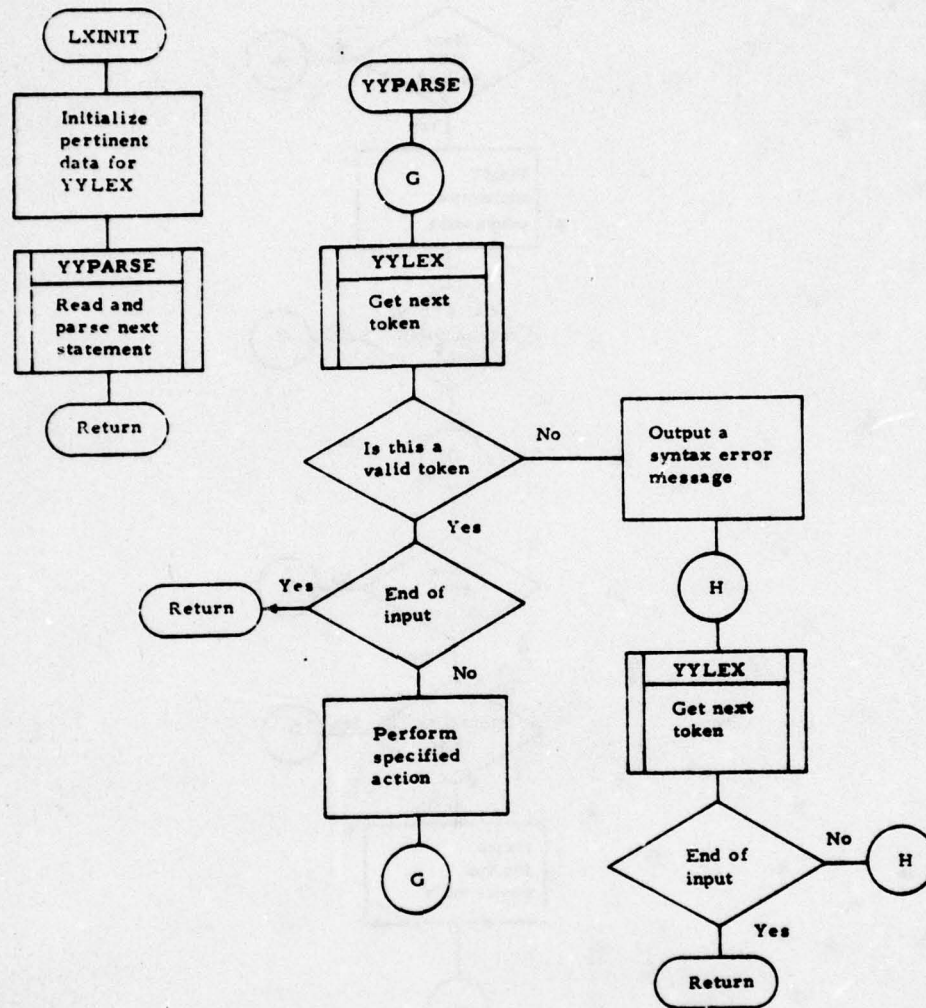


Figure 12. AATERM Process Data Flow (Sheet 18 of 19)

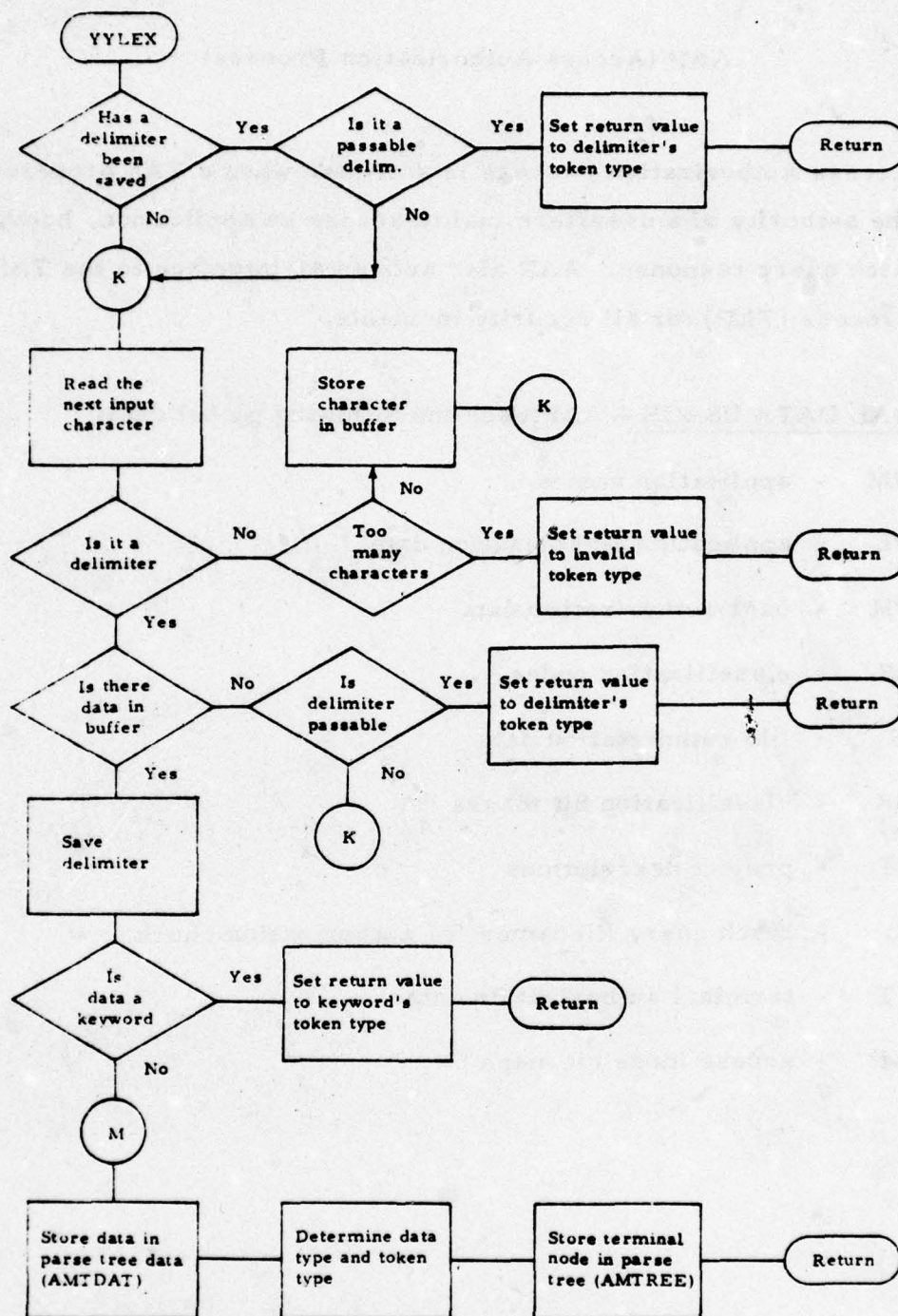


Figure 12. AATERM Process Data Flow (Sheet 19 of 19)

AAP (Access Authorization Process)

The Access Authorization Process is activated when a TAS process wishes to check the authority of a user/terminal to access an application, batch host file, or batch query response. AAP also acts as an interface to the TAS Logging Process (TLP) for all security incidents.

GLOBAL DATA USAGE – AAP uses the following global data:

GAPNM	- application names
GAPPL	- application authorization data
GAUTH	- user authorization data
GCLAS	- classification codes
GFILS	- file authorization data
GMASK	- classification bit masks
GPROJ	- project descriptions
GQFIL	- batch query filenames for authorization check
GTAUT	- terminal authorization data
GAMAP	- access mode bit maps

LOCAL DATA USAGE - AAP uses AASTATUS to return a status to calling processes. These statuses are:

- 0 - TAAAUTH both user and terminal are authorized
- 1 - TAANUSR the user is not authorized
- 2 - TAANTRM the terminal is not authorized
- 3 - TAABTH neither the terminal nor the user is authorized

GENERAL PROCESS FLOW - AAP is activated by the Batch Query Processors (BQPs) and the Interactive System Processors (ISPs) to verify a user's/terminal's authority to access a specified application (DIAOLS, ISSPIC, RYETIP, SOLIS) or files resident at a batch host. It is also activated by the DISPLAY and PRINT commands to verify that the clearance of the user/terminal exceeds or equals the classification of a batch query response. AAP first clears its return status word and reads the user's GAUTH file. The terminal identifier of the user's terminal is determined, and items GTNAP and GTCLR are read from its GTAUT file. If the input function is a batch query response display clearance check, AARDS is called to process the function. Otherwise, both the user's and terminal's GAPPL files are read in and AAP calls AAPAC to process an application access attempt. If filenames in GQFIL are to be authorized and if both terminal and user are authorized for the given application, AAFAC is called to process file access attempts. AAP then returns to the calling process with AASTATUS, status of its authorization check. (See figure 13 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

AAAPAC (Application Access Processor) – AAAPAC performs the specific processing required for checking the authority of a user/terminal to access an application. A search is made through the user's GAPPL file for a match to the input application. If a match is not found, the global AAP return status is set to one (TAANUSR), indicating that the user is not authorized to access the specified application and an error message to that effect is output to the user's terminal. If a match is found for the user, then a search is made through the terminal's GAPPL file for a match to the input application. If a match is not found, the global status word is set to two (TAANTRM) indicating that the terminal is not authorized to access the specified application and an error message to that effect is output to the user's terminal. Before returning, if either the user or terminal is not authorized to access the specified application, AAAPAC invokes the TAS Logging Process (TLP) to record the event.

Note: Although the TASTMASTER has access to all applications, interactive application logon information (GILOG file) must exist for an interactive host in order for proper communication to be established with the host.

AAFAC (File Access Processor) – AAFAC performs the specific processing required for checking the authority of a user/terminal to access a file resident at a batch host. AAFAC reads in the user's and terminal's GFILS files and reads the query filenames in GQFIL, a file built by the activating process. Each filename in GQFIL is processed in the following manner: 1) A search is made through the user's GFILS file for a match on the filename and, if found, on the file access mode request. 2) If a match is not found or if a match is found but the file's access authorization is insufficient, the status byte for that file, GFAUTH, is set to one and the AAP status word,

AASTATUS, is logically "OR"ed with one, indicating that the user is not authorized access to the specified file/access mode. A diagnostic is displayed at the user's screen and the next filename is processed. 3) If the user is authorized to access the file for the given mode, a search is made through the terminal's GFILS file for a match to the filename. 4) If a match is not found or if the file's access authorization is inadequate, the file status byte is set to two and AASTATUS is logically "OR"ed with two, indicating that the terminal is not authorized access to the specified file/access mode. A diagnostic is displayed at the user's screen and the next filename is processed. 5) In any case, the next filename is processed.

After all filenames have been processed, GQFIL is rewritten with the new file status bytes and a check is made to see if all files were authorized to both the user and terminal. If so, AAFAC returns immediately. Otherwise, the global AAP status (AASTATUS) is sent to the TAS Logging Process which is activated to record the security violation. AAFAC returns with a normal status.

AARDS (Batch Response Display Processor) — AARDS performs the specific processing required for checking the clearance of a user/terminal against the classification of a batch query response. First it is determined whether the clearance is a valid clearance code as stored in GCLAS. If not, an error is output, AASTATUS is set to one (TAANUSR) and a return is made without calling TLP. If the user's clearance is less than that required by the response, AASTATUS is set to one (TAANUSR) and an error message is output to the user indicating that he is not authorized to see the response. Otherwise, the terminal's clearance is checked. If the terminal's clearance is less than that required by the response, the global status word is set to two (TAANTRM) and an error message indicating that the terminal is not cleared for the response is output to the user. Before returning, if either the user or the terminal is not cleared for the response, AAFAC activates the TAS Logging Process (TLP) to record the security violation.

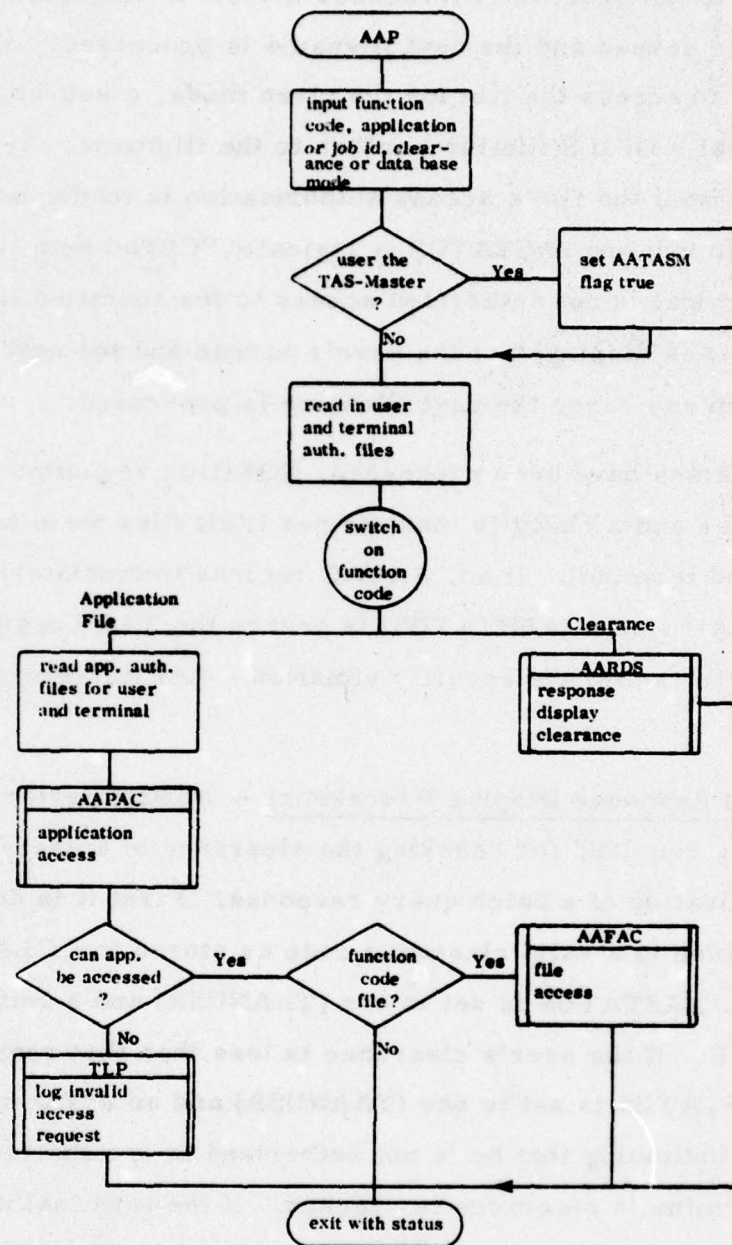


Figure 13. AAP Process Data Flow (Sheet 1 of 4)

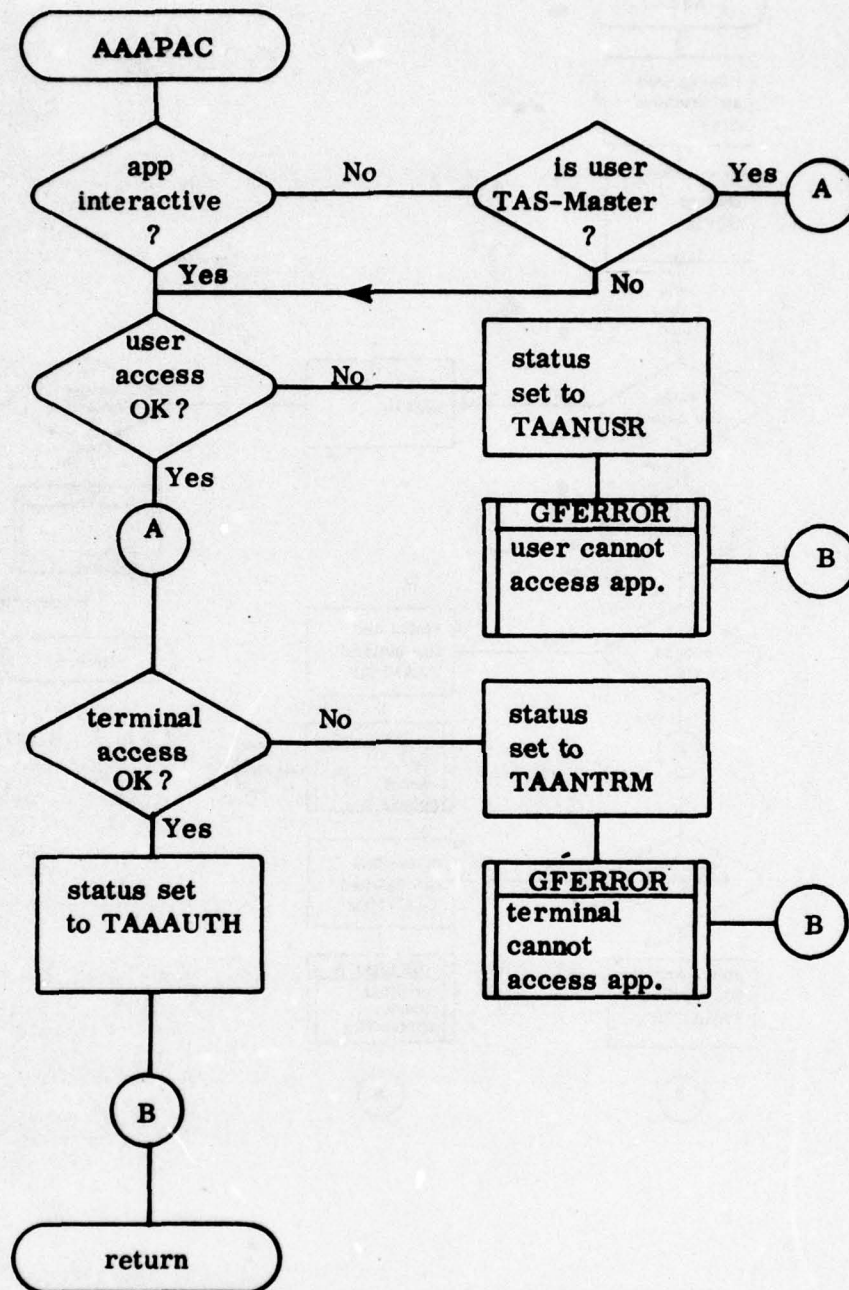


Figure 13. AAP Process Data Flow (Sheet 2 of 4)

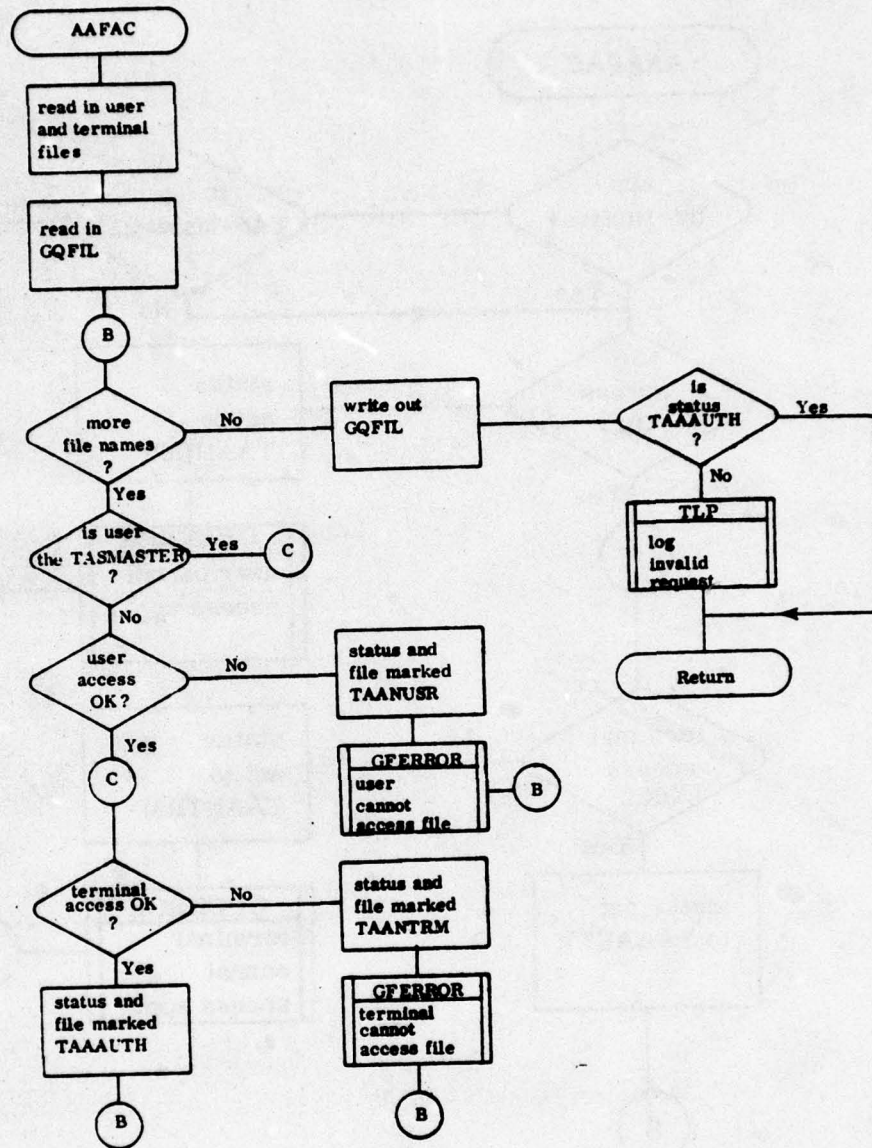


Figure 13. AAP Process Data Flow (Sheet 3 of 4)

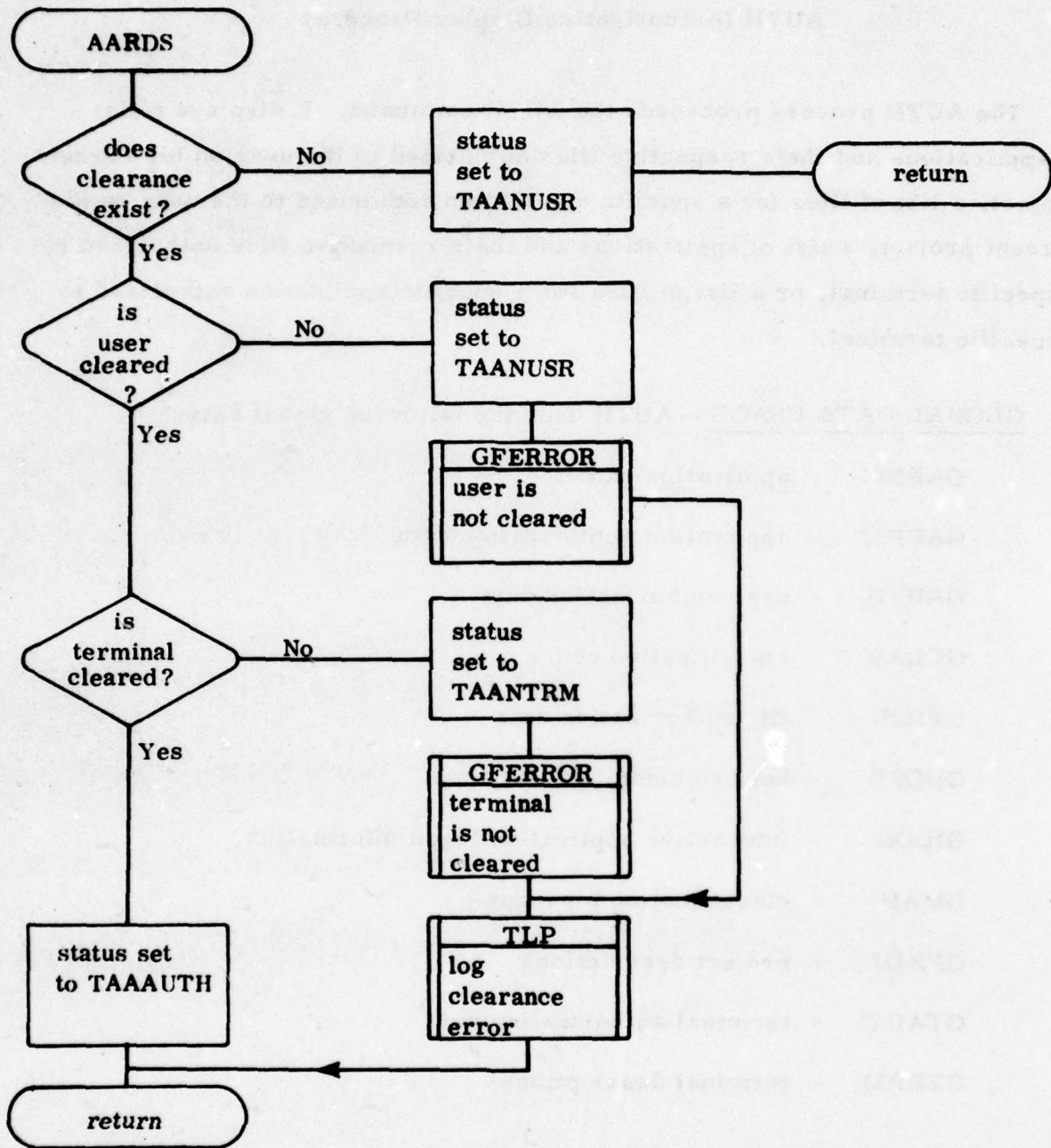


Figure 13. AAP Process Data Flow (Sheet 4 of 4)

AUTH (Authorization Display Process)

The AUTH process processes the AUTH command. It displays a list of applications and their respective files authorized to the user on his current project, a list of files for a specific application authorized to the user on his current project, a list of applications and their respective files authorized to a specific terminal, or a list of files for a specific application authorized to a specific terminal.

GLOBAL DATA USAGE - AUTH uses the following global data:

GAPNM	- application names
GAPPL	- application authorization data
GAUTH	- user authorization data
GCLAS	- classification codes
GFILS	- file authorization data
GHOST	- host statuses
GILOG	- interactive application logon information
GMAP	- classification bit maps
GPROJ	- project descriptions
GTAUT	- terminal authorization data
TERM	- terminal descriptions

LOCAL DATA USAGE - AUBUF is a character array used by AUTH to construct each print line. AUAPTST is a flag which is true if a specific application is requested as an input parameter and is false if no application is input.

GENERAL PROCESS FLOW - If the user who initiated the AUTH command had not input any arguments, the AUTH process assumes that the user is requesting a list of applications and their respective files which are authorized for this current project. Function AUSER is called to read in the appropriate user authorization files, GAUTH and GPROJ. In turn, function AUWAPP reads in the GAPPL and GFILS files and displays applications and their batch files/access authorization information. After all requested information is output, AUTH terminates.

If, however, the user has input one or more parameters, AUTH first determines whether or not the first input is a terminal identifier, found in GTERM. When the input is a valid terminal identifier, the terminal's GTAUT file is read, and the number of applications assigned for the terminal is retrieved. If a second parameter is input, AUTH sets the AUAPTST flag to true and calls AUWAPP to search the terminal's GAPPL files for that application name. If the application name is not located in GAPPL, a diagnostic is output and AUTH terminates. Otherwise, for each application whose files are to be displayed, function AUWAPP outputs the application name, and if the application is a batch host, AUWAPP picks up the number of authorized files, and reads in that application's GFILS file for the specified terminal. For each authorized file, the file name and access mode are output. After all filenames for all specified applications have been displayed, AUTH terminates.

If the first parameter is not a terminal identifier, AUTH calls function AUSER to read in the appropriate user authorization files, GAUTH and GPROJ.

Since one parameter has been input with the AUTH command, AUTH sets the AUAPTST flag to true and calls AUWAPP to search the GAPNM (application names) file validating that the input is indeed a valid application name. If the application name is not valid, an error message is output and AUTH terminates; otherwise, the AUWAPP function displays the application name and the access authorization information for the associated batch files. Having listed all files names for the specified application, AUTH terminates. (See figure 14 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

AUSER (User's Authorization Files) – Initially AUSER checks that the user is not the TASMASTER since the TASMASTER has access to all applications and all files for batch hosts. If the user is the TASMASTER, an error is output and AUTH terminates. Otherwise, AUSER reads the GAUTH and GPROJ files of the user, prepares the pathnames for GAPPL and GFILS, and stores the number of applications for the current project. AUSER returns to the calling function.

AUWAPP (Write Applications) – The purpose of the AUWAPP function is to output application and file authorization data for both terminals and users. File GAPPL is read and each application is processed by the following procedure: 1) If all applications are desired or if the application name matches the input application name, the application name is displayed. 2) If the application is a batch host, GFILS is read and the list of file names and access codes are displayed for that application. 3) If a specified application's data has been output, AUTH terminates; otherwise, the next application located in GAPPL is processed as described.

If all the applications have been processed and the requested application is not found, a user diagnostic is output and AUTH terminates. Whenever a default project or terminal does not have an application defined for it, an appropriate header and message is output and AUTH terminates. Under normal conditions, however, function AUWAPP returns to the calling function.

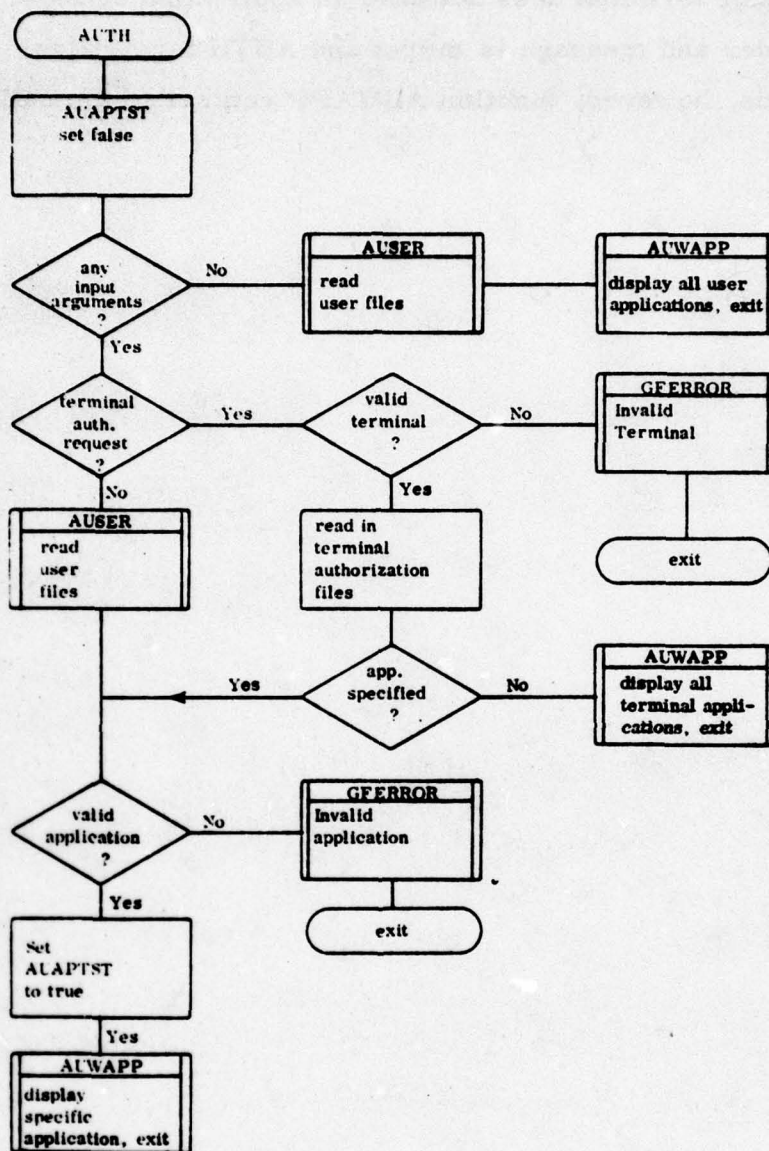


Figure 14. AUTH Process Data Flow (Sheet 1 of 3)

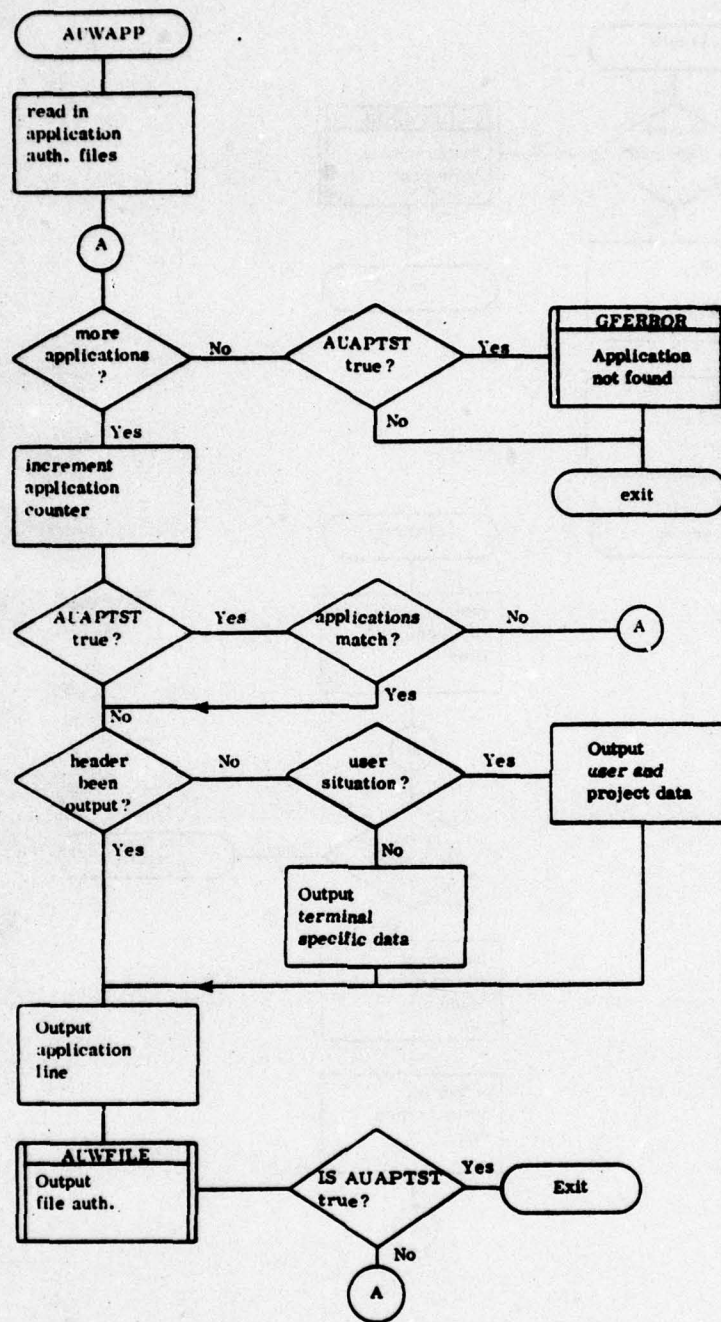


Figure 14. AUTH Process Data Flow (Sheet 2 of 3)

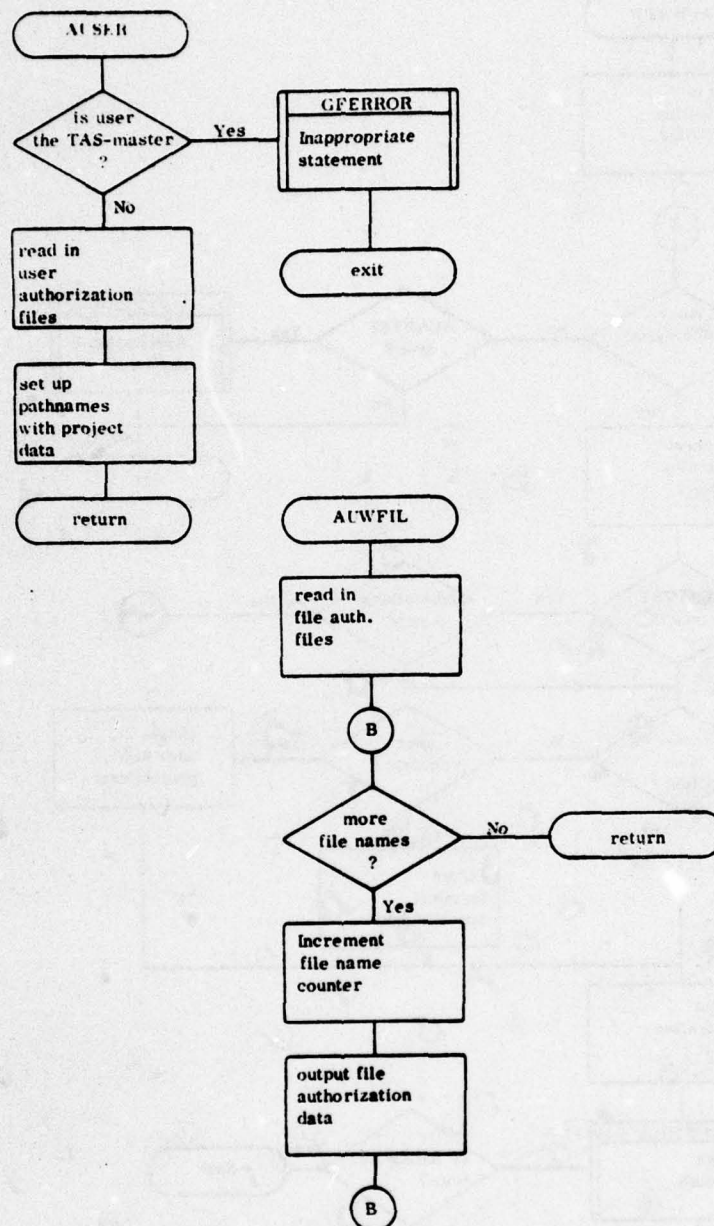


Figure 14. AUTH Process Data Flow (Sheet 3 of 3)

BQP (Batch Query Processor)

The Batch Query Processors (BQPs), i.e., DIAOLS, RYETIP, and ISSPIC, process batch query requests. A BQP accepts either a file or terminal input, searches for filenames and keywords in the input and validates that the user and terminal are allowed access to the application and named files. Authorized queries are sent to the appropriate application.

GLOBAL DATA USAGE - BQPs use the following global data:

- GFILE - user text file descriptions
- GBQFN - user batch query filenames
- GBQST - user batch query statuses
- GQFIL - batch query filenames for authorization check

LOCAL DATA USAGE - The BQPs' significant, locally defined variables and arrays are:

- BQHOST - a pointer to the host ID in ASCII form.
- BQJBID - a pointer to the JOBID for this query. The first character in the JOBID reflects which BQP is processing the query.
- BQKYWRD - a two-dimensional array which contains the host's verbs (keywords) which call out or reference files.

BQVRBTYP - an array where the verb types (access modes) of the verbs in BQKYWRD are stored. BQKYWRD and BQVRBTYP are parallel.

GENERAL PROCESS FLOW - There is a BQP for each batch application on the network; each bearing the name of the application. At present, RYETIP, DIAOLS, and ISSPIC are the TAS batch hosts. The BQP calls the Access Authorization Process (AAP) to determine whether the user at his current terminal can access the specified application. If not, then a diagnostic is output and the BQP process terminates. If the user can access the application, the BQP determines whether there is an input parameter. If there is, BQP calls the BQPEDIT function with the specified filename. Otherwise, the BQP calls the BQPEDIT function with no filename. BQPEDIT will verify the existence of a specified filename. At this point, a subset of the editor functions is available to the user. When the user is ready to send his query, he types an 'x' to indicate transmit.

Next the BQP calls BQSCAN to scan the query for keywords as stored in BQKYWRD. When keywords calling out or referencing filenames are found, then BQSCAN also picks up the filenames and stores them in the batch query filenames for authorization check (GQFIL) file. The keyword's access mode (retrieve or update) as stored in BQVRBTYP is stored with the filename in GQFIL. After the scanning is accomplished, BQP calls AAP to determine whether the user at his current terminal can access the named files with the specified access modes. If either the user or the terminal cannot access one or more of the files, a diagnostic is output for each disallowed access, and the editor functions are again made available to the user to modify his query.

If there are no access problems, the BQBQRD function is called to establish a JOBID for the batch query in which the user's ID is inserted. BQBQRD also calls the Batch Query and Response Dispatcher (BQRD) with the JOBID, function type (INTG), application name, pathname for the query (edit) file, and pathname for the response processor for batch query responses (RESPONSE). After the BQBQRD function returns, BQP locks and reads the user's GBQST file, and creates an entry containing JOBID, application ID, date and time of entry to TAS, and a status of 'pending'.

BQP now reads in the first item of the GBQFN file which is the pointer to the last used slot in GBQFN. Its value is stored as a pointer in GBQST and GBQST is written out. The function BQFILE stores the filenames as they appear in GBQFN and writes out GBQFN. The file GQFIL is unlinked at this time. The edit file is copied into a query file with the JOBID as its name, and the file GBQST is unlocked via GFUNLOCK.

The BQP outputs the JOBID to the user and calls BQPEDIT again, i. e., returns to the point at which the editor functions are available to the user. The BQP is terminated by the user typing 'q', the quit command. (See figure 15 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

The BQP program description in effect is describing the general program flow for the three current batch query processors DIAOLS, ISSPIC and RYETIP. For the sake of generality, the BQP function names are prefixed by 'BQ' instead of 'DI', 'IS' or 'RY' as they would actually appear in program listings.

BQSCAN - The BQSCAN function creates the GQFIL file and opens the user's input (edit) file. A scan is made of the input file searching for verbs that reference files. Whenever a word is a valid verb (as stored in array

BQKYWRD), the function BQGETF locates and stores the filename which is referenced.

BQGETF — The BQGETF function determines whether a verb references a file. If the verb does not reference a filename, BQGETF returns a normal value. If the verb does not specify a file, but does reference a filename, the last mentioned filename and the verb type (from BQVRBTYP) are stored in GQFIL. BQGETF returns a normal value to the calling function.

If the verb does specify a file, a filename is located in the user's input statement. If the filename/verb type is not already in GQFIL, the name/verb type is stored. The input statement is further searched for other filenames and, if found, the filenames are processed as above. The function BQGETF returns a positive value to the calling function.

BQBQRD — The BQBQRD function calls the TAS global function GFJOBID to get the next JOBID to be assigned to the user's batch query. The user ID is inserted into the JOBID and a fork/execute sequence is performed to call the Batch Query and Response Dispatcher (BQRD) to send the batch query through the network. BQBQRD then returns to the calling function.

BQFILE — The first item of GBQFN contains the pointer to the last used entry in GBQFN. BQFILE increments this pointer by the number of files in GQFIL. This first item of GBQFN is then written out. The file count is stored in the first item of GQFIL and the contents of GQFIL are appended to the GBQFN file. BQFILE then returns to the calling function.

BQPEDIT (Create and/or Edit A Query File) – The BQPEDIT functions provide the capability for a TAS user to create and/or edit a query file while in a Batch Query Processor (DIAOLS, ISSPIC or RYETIP). BQPEDIT is essentially the UNIX text editor "ed" with the following modifications:

1. The 'f', 'k', 'l', 'w' and '!' commands have been removed.
2. A prompt character '*' is output when BQPEDIT is ready for a command.
3. Filenames may consist only of letters, digits and '.' and may be from one to thirteen characters long.
4. An 'x'mit command has been added to allow the edited query to be transmitted across the network to the specified batch application.

The remainder of BQPEDIT operates exactly as does "ed"; consequently, the areas addressed by this description pertain only to the modifications made to "ed" to make it satisfactory for use under TAS.

The following processing is done by BQPEDIT to allow the capabilities of "ed" to be used by TAS:

During program initialization, the global file GFILE is opened.

Prior to the execution of each command, a prompt character '*' is displayed.

When an 'e'dit or 'r'ead command is entered, GFILE is searched for an entry with a filename matching the one referenced in the command. If a match is not found, a '?' is output and the command is ignored. Otherwise, the index into GFILE for the file is saved for later reference.

When an 'x'mit command is entered, the edited query is written out to the file "current_file" in the user's directory. This file is copied by BQRD to be used in transmitting the query to the specified batch application. BQPEDIT then returns to the calling function.

When a 'quit command is entered, BQPEDIT exits, thus terminating the BQP.

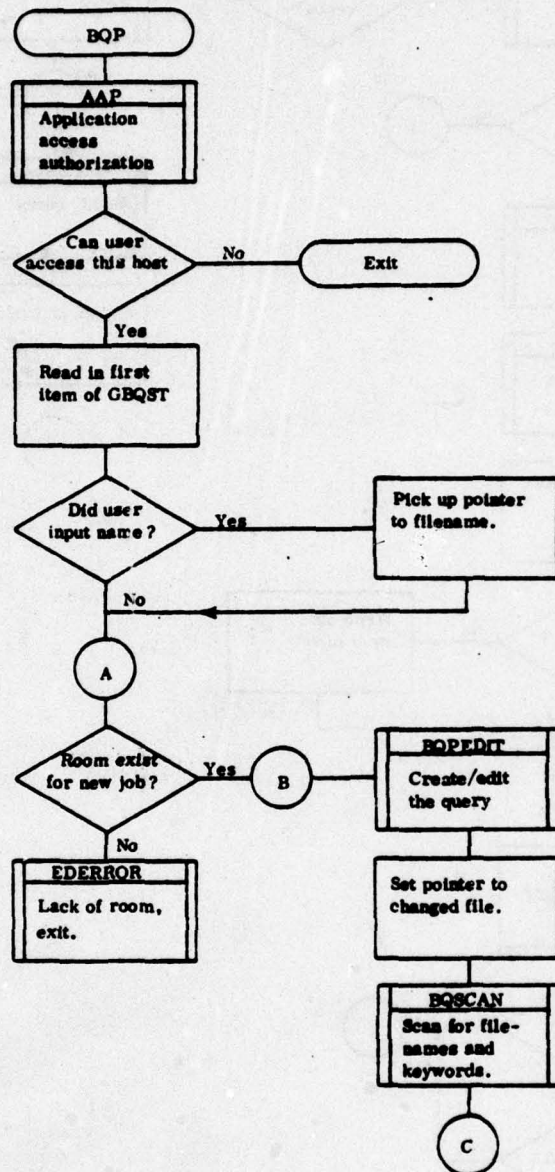


Figure 15. BQP Process Data Flow (Sheet 1 of 15)

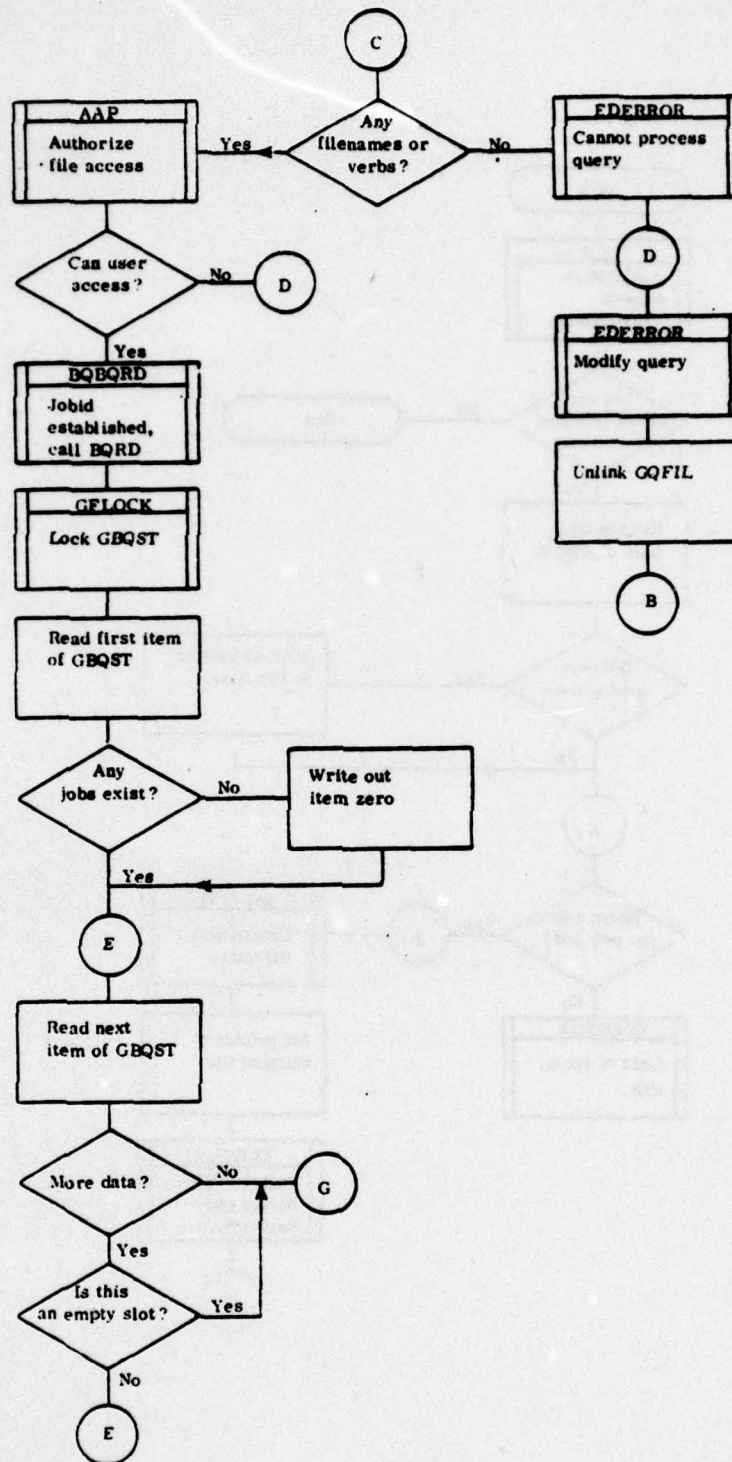


Figure 15. BQP Process Data Flow (Sheet 2 of 15)

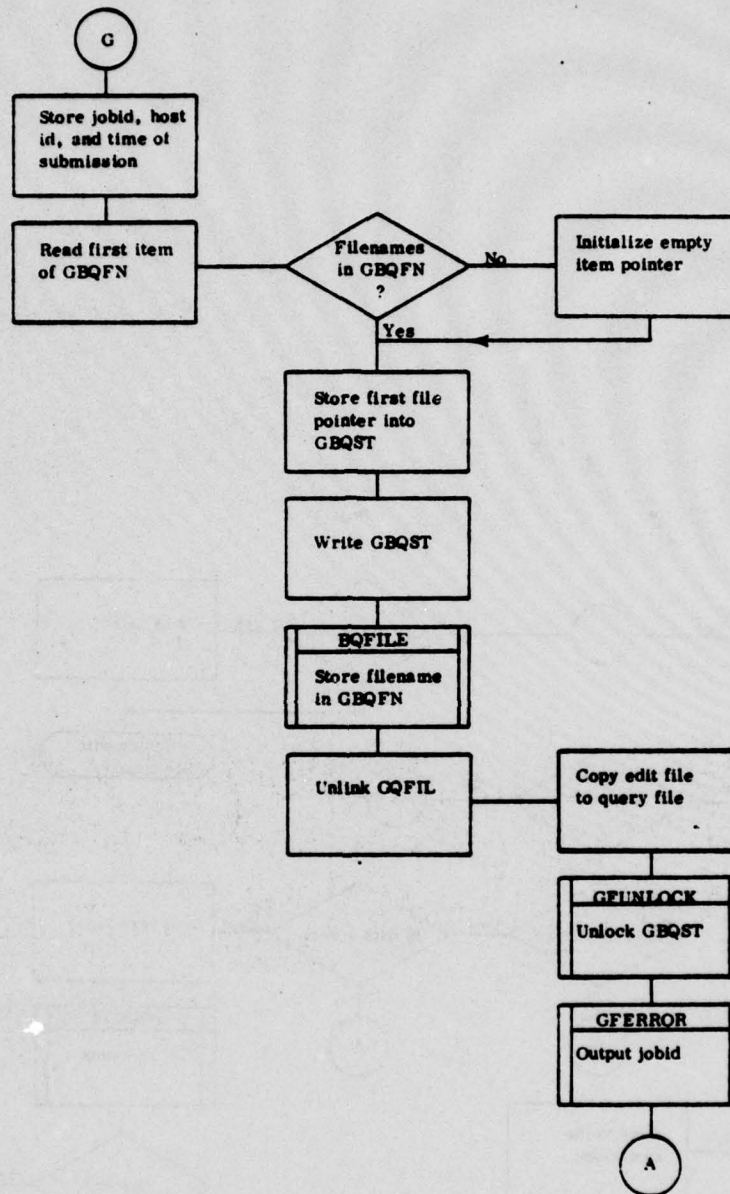


Figure 15. BQP Process Data Flow (Sheet 3 of 15)

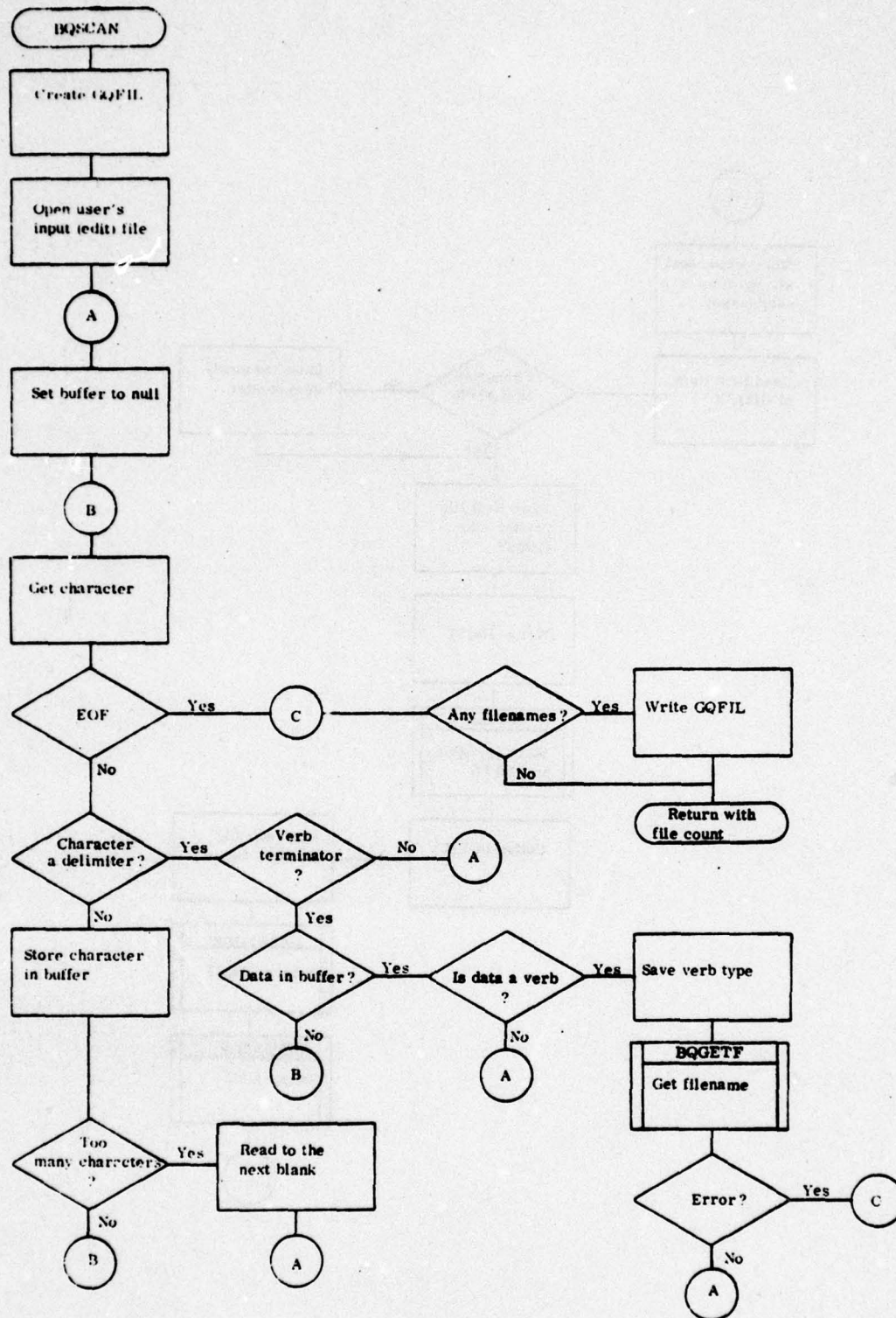


Figure 15. BQP Process Data Flow (Sheet 4 of 15)

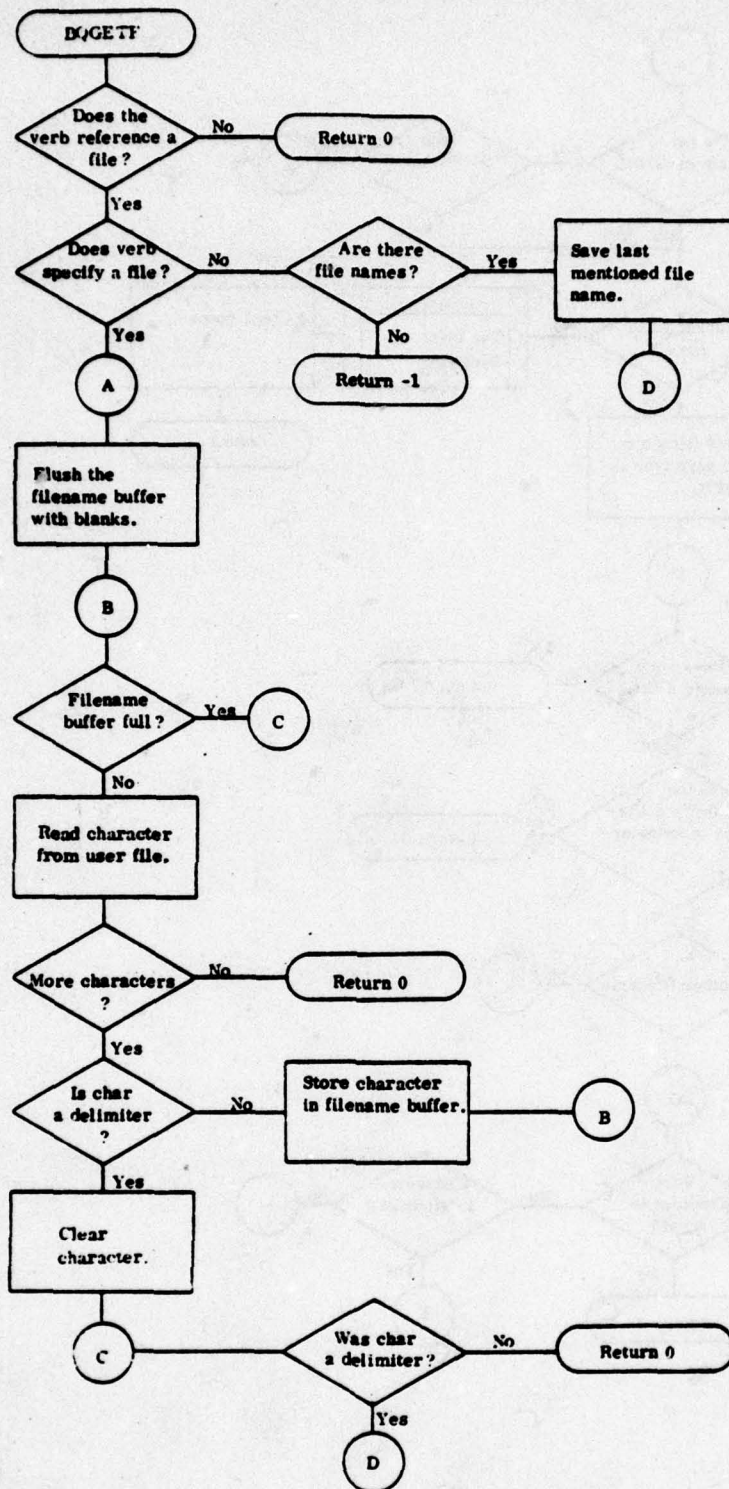


Figure 15. BQP Process Data Flow (Sheet 5 of 15)

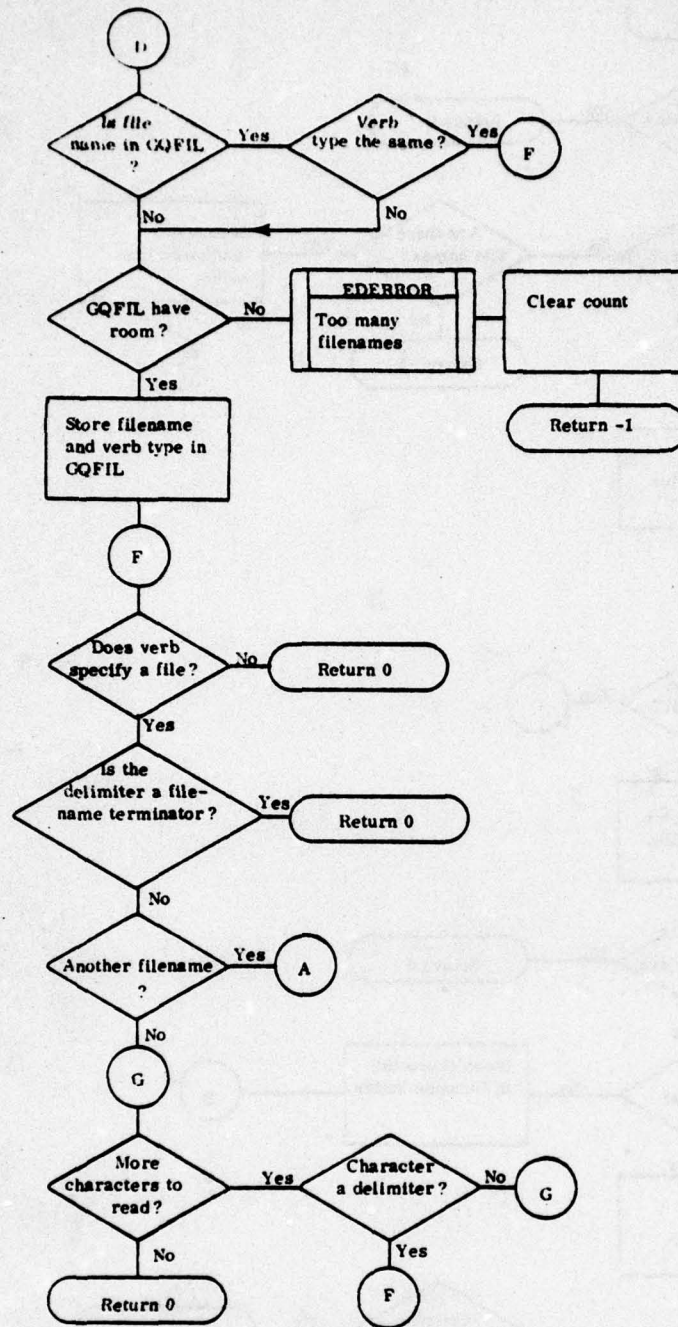


Figure 15. BQP Process Data Flow (Sheet 6 of 15)

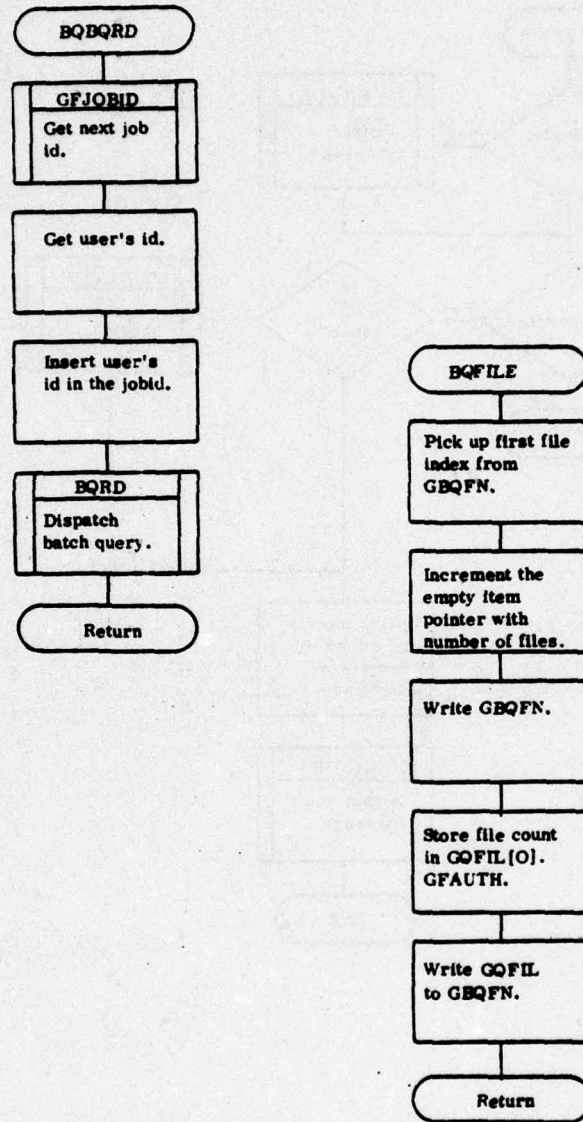


Figure 15. BQP Process Data Flow (Sheet 7 of 15)

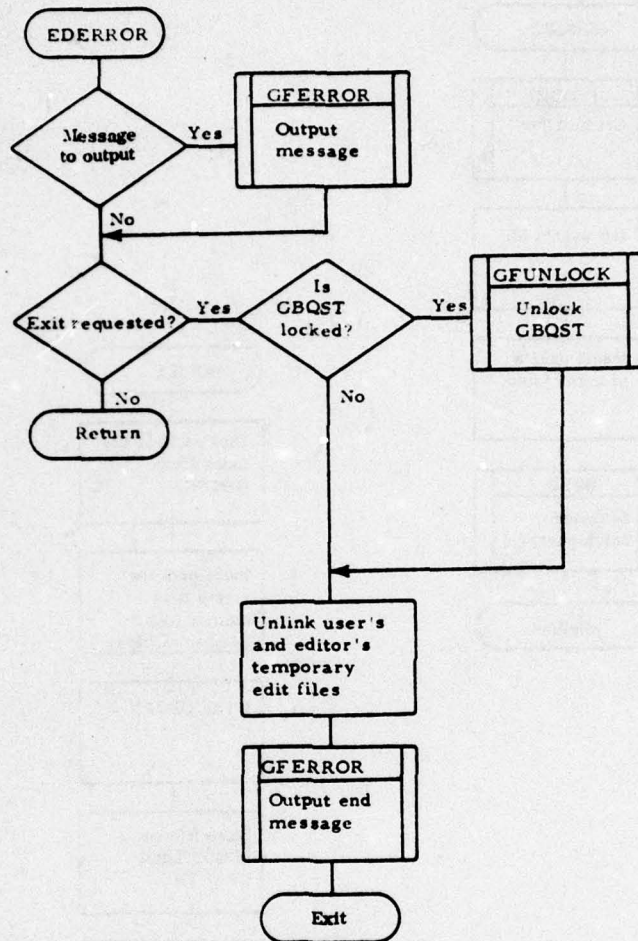


Figure 15. BQP Process Data Flow (Sheet 8 of 15)

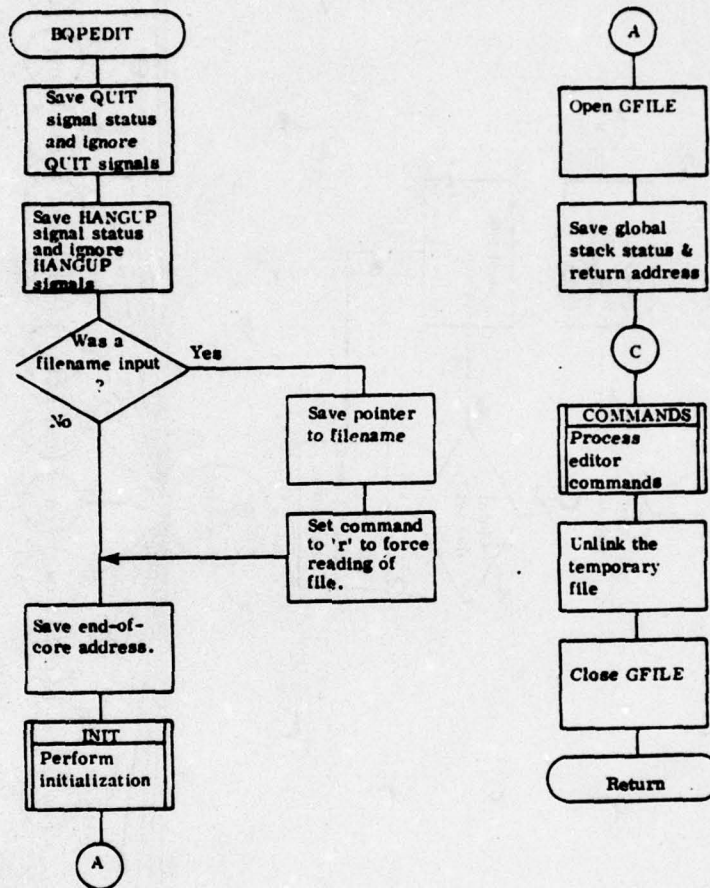


Figure 15. BQP Process Data Flow (Sheet 9 of 15)

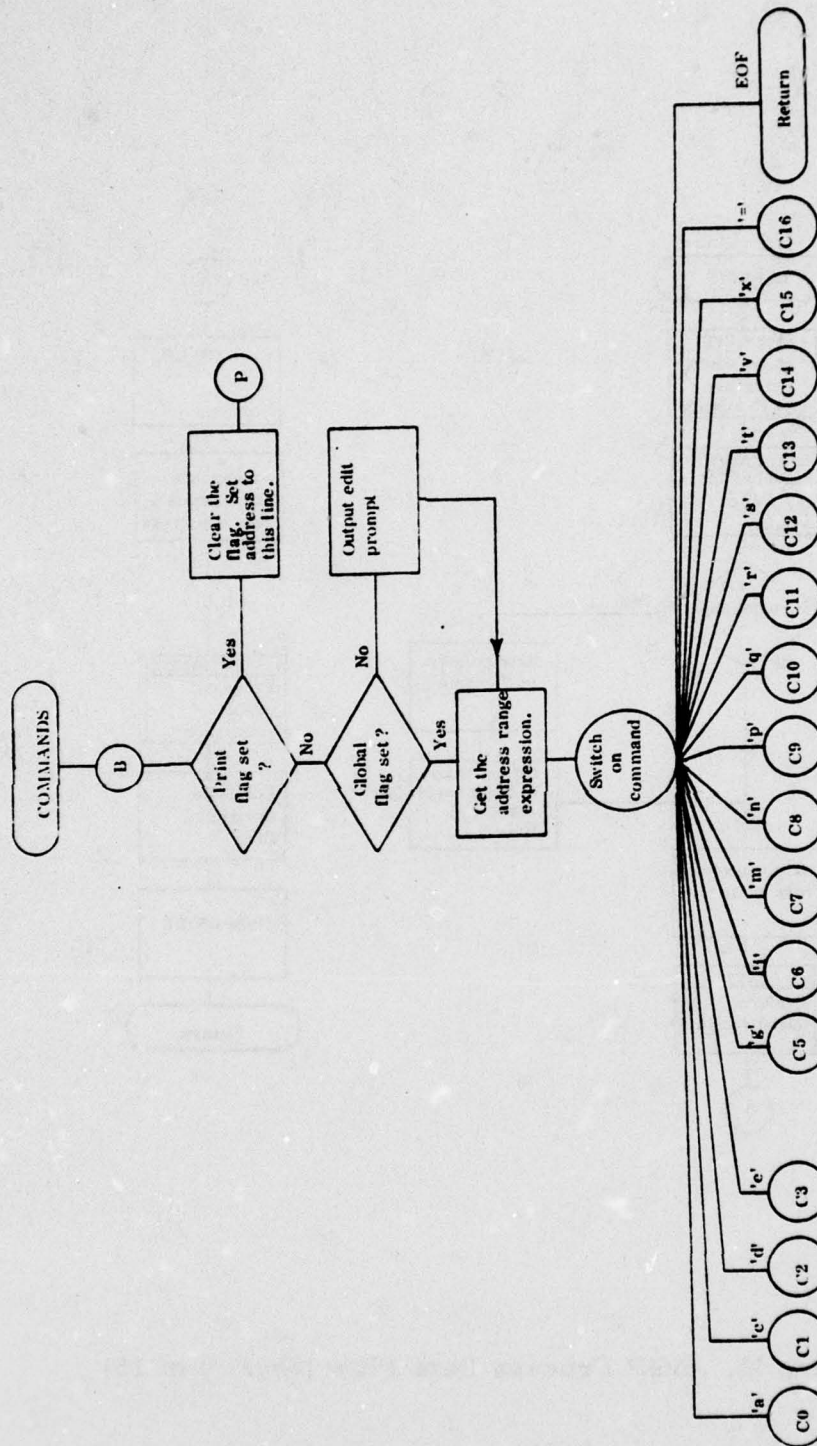


Figure 15. BQP Process Data Flow (Sheet 10 of 15)

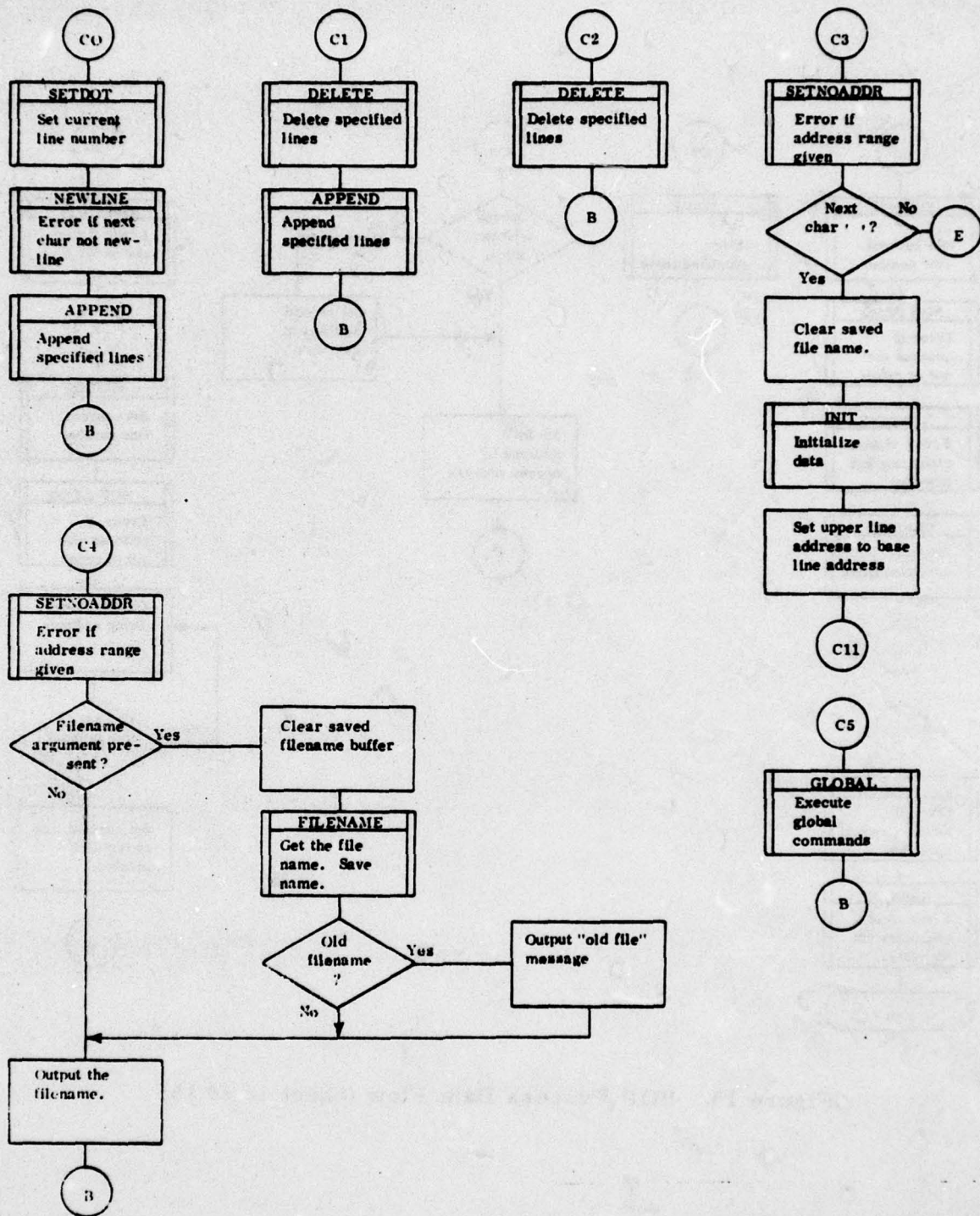


Figure 15. BQP Process Data Flow (Sheet 14 of 15)

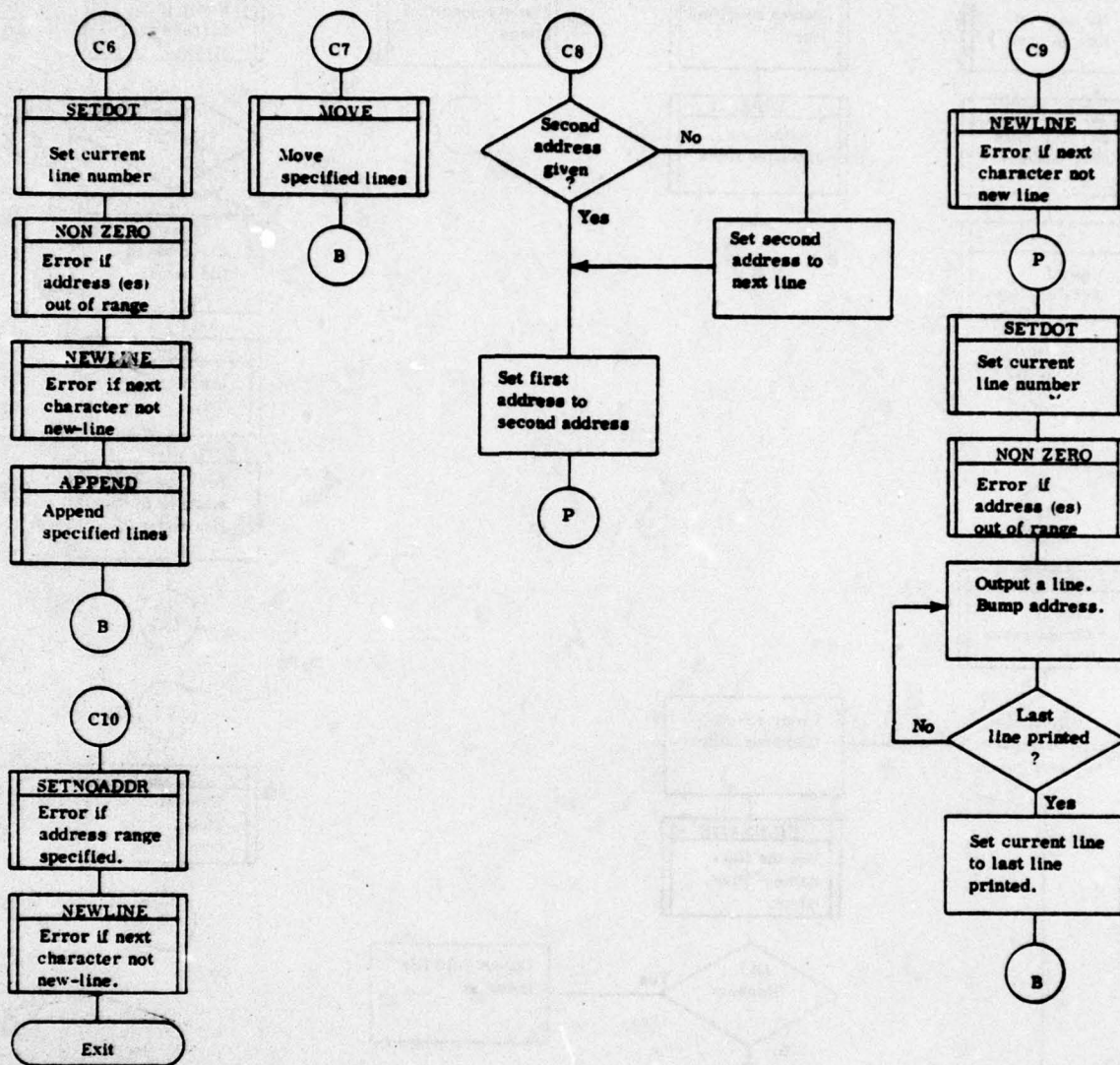


Figure 15. BQP Process Data Flow (Sheet 12 of 15)

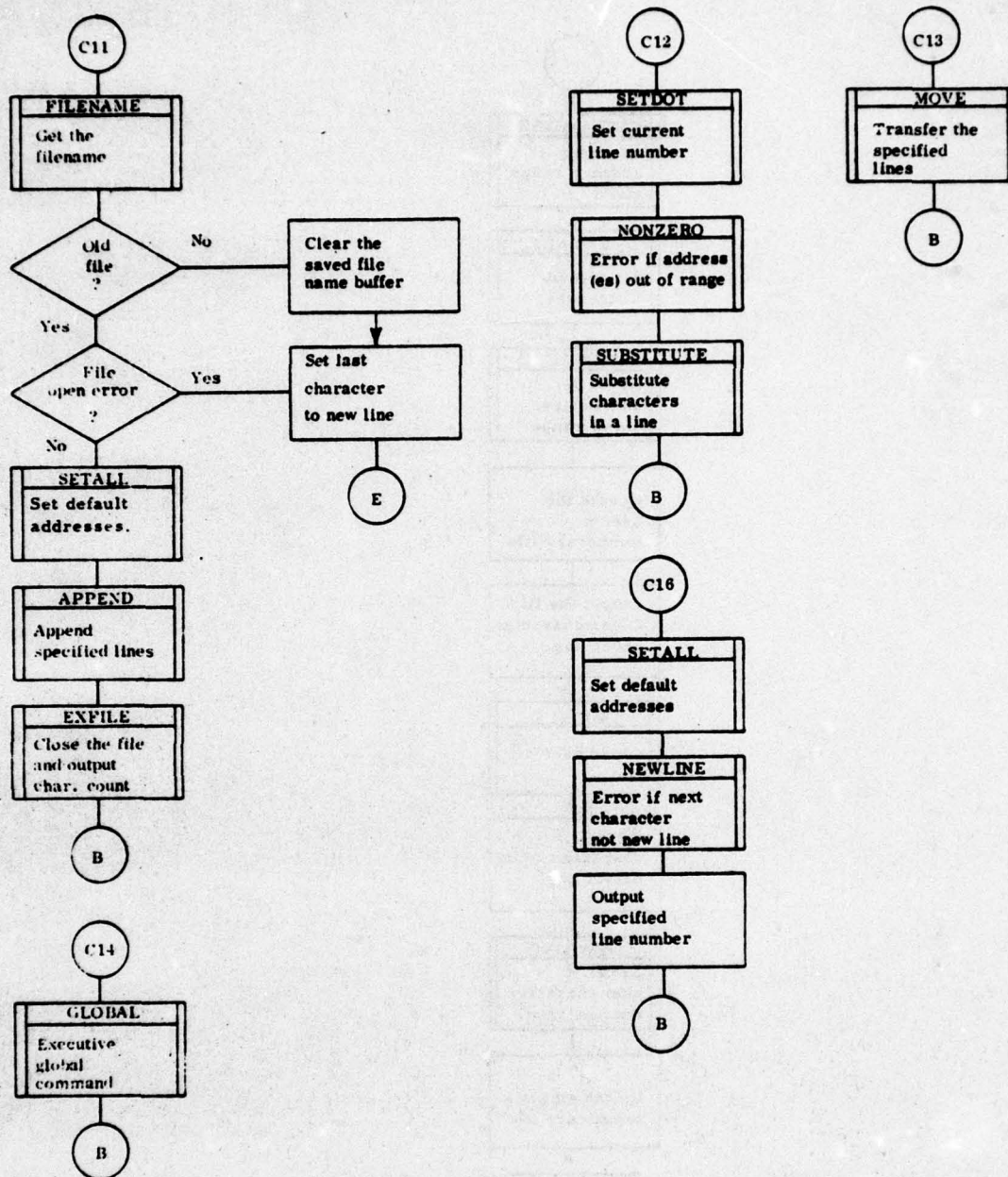


Figure 15. BQP Process Data Flow (Sheet 13 of 15)

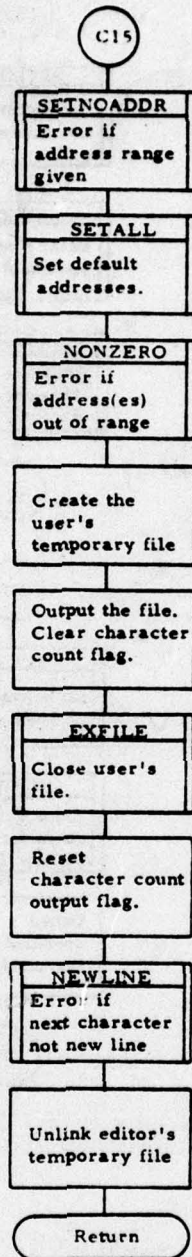


Figure 15. BQP Process Data Flow (Sheet 14 of 15)

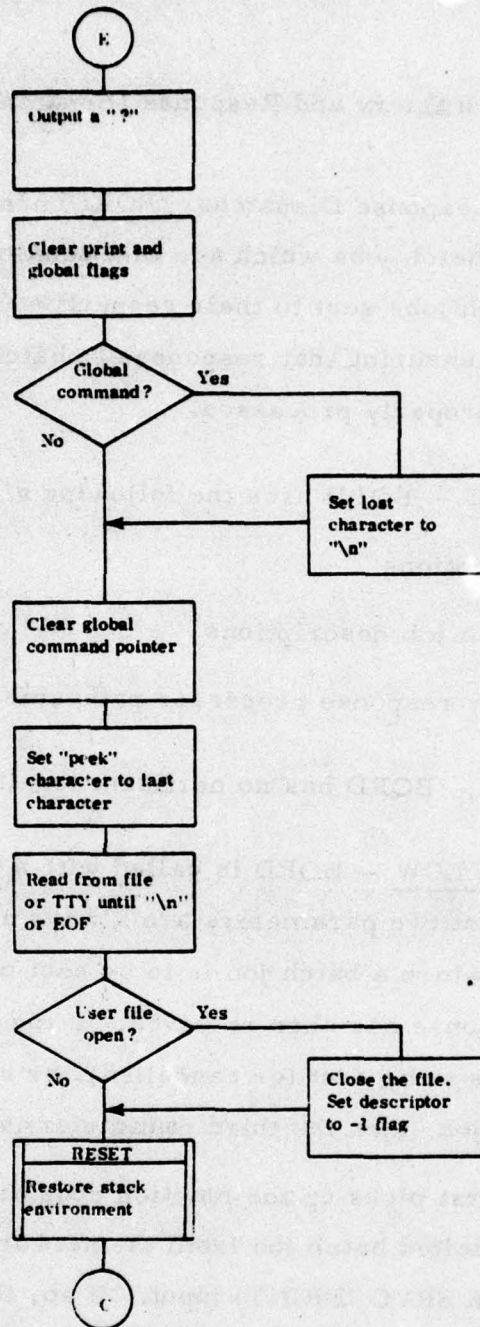


Figure 15. BQP Process Data Flow (Sheet 15 of 15)

BQRD (Batch Query and Response Dispatcher)

The Batch Query and Response Dispatcher (BQRD) process is responsible for maintaining a queue of batch jobs which are in a pending state. BQRD is responsible for having batch jobs sent to their respective hosts, for resending them if necessary, and for ensuring that responses to batch jobs and unsolicited batch messages are properly processed.

GLOBAL DATA USAGE — BQRD uses the following global data:

GHOST - host descriptions

GJOBS - BQRD batch job descriptions

GRES P - batch query response processor pathnames

LOCAL DATA USAGE — BQRD has no pertinent local data.

GENERAL PROCESS FLOW — BQRD is called with a varying number of input parameters. The first two parameters are always a function code and an application (host) ID to which a batch job is to be sent or from which a batch job or batch job response has been received. If the function code specifies that a batch job is to be sent (or cancelled), or specifies a response to a previously sent batch job, then the third parameter is always a JOBID.

When called, BQRD first picks up the function code and host ID. If the function specifies an unsolicited batch job from the network, then BQRD checks to see if the job is a SRVC (PRINT) input. If so, then the pathname of the network input file (containing the print data), the function code, and the host ID are picked up. BQRD then calls the SRVC Response Processor (SRESP) with the above data via BQCALLR and exits via BQERROR. If the

input is other than a SRVC (PRINT), then it reflects a change in a host's status. These functions are SRVC (READY, BREAK, ALARM), and network read error.

To process a change in a host's status, BQRD first locks the Host Descriptions (GHOST) file and reads it. If the function specifies a network read error, then the function is changed to indicate a SRVC (BREAK) or host down. If the host ID/status specify TAS/host down (obviously not applicable to unsolicited network inputs), then BQRD checks to see if any other host's status still reflects an up or READY state. If the host ID is not TAS or the host ID is TAS and no other hosts are up, then the specified host's status in GHOST is changed to the specified status (function). BQRD writes GHOST and then calls GFUNLOCK to unlock GHOST. If the host ID is TAS, BQRD calls BQERROR to exit. Otherwise, BQRD picks up the function code and the host ID and calls SRESP via BQCALLR. If the function is SRVC (READY) or host up, then BQRD calls BQPQP to process pending queries. BQRD then calls BQERROR to exit.

If the function code does not specify an unsolicited network input, then BQRD locks the specified host's BQRD batch job descriptions (GJOBS) file. BQRD then determines whether the function specifies an input to the network, a response from the network, cancel or lost. If the function specifies input to the network (INTG, SRVC(PRINT, TRACE, CHECK, READY, BREAK)), then BQRD checks to see if the function specifies automatic traces. In this case, there is no third parameter of a JOBID as BQRD itself generates the JOBIDs for the SRVC(TRACE)s. If automatic traces are to be sent, then BQRD calls BQTRACE to send the traces and calls BQERROR to exit. If the function specifies a SRVC(TRACE) on a particular JOBID, then BQRD reads the job's data file to obtain the JOBID to trace. BQRD searches the GJOBS file for the specified JOBID. If the JOBID is

not present, indicating that either the JOBID is invalid or the specified INTG is no longer pending, then BQERROR is called to exit with an error value. If the trace can be sent or the function is other than a SRVC(TRACE), then BQRD searches GJOBS for an empty item and stores the job's data in GJOBS (function code, status of pending, submission time, JOBID, and traced JOBID if a SRVC(TRACE)). If the job has a response processor to be called, then BQSTRES is called to store its pathname in the batch query response processor pathnames (GRES P) file. BQRD locks GHOST via GFLOCK and then reads GHOST. If the function specifies an INTG, then the query counters for the specified host and for TAS in GHOST are incremented. BQRD then writes GHOST if it was modified and unlocks GHOST via GFUNLOCK. If the function does not specify a SRVC (READY, BREAK or CHECK), then BQRD checks GHOST to see if the status of either TAS or the receiving host is down. If either host is down, then BQRD does not attempt to send the job, but goes to the TAS Logging Process (TLP) switch processing as specified later. If both TAS and the specified host are up and/or the job is a READY, BREAK or CHECK, then function BQNET calls INI to attempt to send the job through the network. If the job was successfully sent through the network, then BQRD goes to the TLP switch processing described later. If the job was not sent through the network, and the job is other than a READY, BREAK, or CHECK, then BQRD also goes to the TLP switch processing. However, if a READY, BREAK, or CHECK was not successfully sent through the network, then BQRD goes to its cleanup processing as specified at the end of this description.

If the function specifies neither an unsolicited network input nor an input to the network, then BQRD must process either a cancel request from a user, a network response to an existing job, or a lost INTG (an internal function code). If the function is a network response and there is an input response data file, then BQRD saves the pathname of the data file. If the

function is a SRVC (TRACK), a response to a SRVC (TRACE), then BQRD searches GJOBS to see if the SRVC (TRACE) for the specified JOBID is still present. (Automatic traces are deleted from GJOBS by the Pending Queries Processor, function BQPQP). If the SRVC (TRACE) is still pending, then BQRD goes to the network response switch processing described below. If the SRVC (TRACE) is not found, then BQRD goes to its cleanup processing described at the end of this section. If the function is other than a SRVC (TRACK), then BQRD searches GJOBS to see if the referenced JOBID is still pending or, in fact, if it exists. If the JOBID is not found and the function code is cancel, then BQRD performs a fork/execute sequence to call TLP with a cancel JOBID and BQRD then goes to its cleanup processing described at the end of this section. If the JOBID is not found and the function specifies lost INTG, then BQRD goes to its cleanup processing. If the JOBID is not found and the function is other than cancel or lost, then BQRD performs a fork/execute sequence to call TLP with a nonexistent JOBID and BQRD then goes to its cleanup processing as described at the end of this section. If the referenced JOBID is still pending, then BQRD checks to see if the function code implies the completion of an INTG. If it does, then BQRD locks GHOST via GFLOCK and reads GHOST.

BQRD switches on the function code for network response processing. If the function code is RLSE, then BQRD performs a fork/execute sequence to call TLP with a RLSE received. If the INTG status is set to cancelled, then BQRD goes to the cleanup processing described at the end of this section. Otherwise BQRD sets the job's status in GJOBS to RLSE received and goes to the TLP switch processing as described later.

If the function code is SRVC (TRACK), then BQRD checks the trace result to see if the job is still active. If the trace result is 'NO RECORD', then BQRD sets a lost INTG flag if the trace was user generated and sets

an INTG retransmit flag if the trace was an automatic trace. The trace result and the traced JOBID are picked up to be sent to the response processor (if any), and BQRD goes to the TLP switch processing described later.

If the function is a RCPT for a SRVC (PRINT, CHECK, READY, or BREAK), then BQRD checks to see if there is a response processor to be called. If there is, then BQDLRES is called to pick up the response processor pathname and delete it from GRESP. The GJOBS item for the job is deleted by being set to zeroes. The function code is saved and, if the job RCPTed is a SRVC (CHECK), the host ID is saved. BQRD then goes to the TLP switch processing described later.

If the function is an ANSR, then BQRD saves the classification of the answer. Also, and if the function is an ABRT (FAULT or CAUSE), BQRD stores the submission and response times in GHOST for the specified host. After this, for both ANSR and ABRT, and if the function specifies lost INTG or logged out INTG, the function code and JOBID are saved for the response processor. BQRD then decrements the query counters in GHOST for the above functions and also for a cancel function where the INTG was not sent through the network. For all of the above functions, BQRD continues processing as follows. GHOST is written and then unlocked via GFUNLOCK. BQRD checks to see if there is a response processor to be called. If there is, BQDLRES is called to pick up the response processor pathname and delete it from GRESP. The GJOBS item for the job is deleted by being set to zeroes. BQRD then goes to the TLP switch processing as described later.

If the function is a cancel, then BQRD saves the job's current status and sets the job's status in GJOBS to cancelled. BQDLRES is called to delete the job's response processor pathname. If the job's previous status indicates that it had been sent through the network, then BQRD goes to the TLP switch processing described below. If the INTG had not been sent through the

network, then BQRD continues processing as specified in the preceding paragraph by first decrementing the query counters in GHOST.

If the function specifies a logged out message, then BQRD first saves the job type. If the logged out job is an INTG, then BQRD continues processing as described above by first saving the function code and JOBID for the response processor. If the logged out message is a SRVC (TRACE), then BQRD saves the traced JOBID for the response processor. If the logged out message is a SRVC (CHECK), then the host ID is saved for the response processor. BQRD checks to see if there is a response processor to be called. If there is, BQDLRES is called to pick up the response processor pathname and delete it from GRESP. The GJOBS item for the job is deleted by being set to zeroes. A modified function code is saved for the response processor and BQRD then goes to the TLP switch processing.

BQRD switches on the function code for TLP processing. If the function code is INTG and the INTG had not been sent before, then BQRD performs a fork/execute to call TLP with a batch query initialized. If the function code is logged out message and the job type is INTG, then BQRD performs a fork/execute to call TLP with an INTG logged out (no RCPT received). If the function code is lost INTG, then BQRD performs a fork/execute to call TLP with a lost INTG. If the function code is ANSR, ABRT (FAULT or CAUSE), then BQRD performs a fork/execute to call TLP with an INTG response received. In addition, if the job's status is cancelled, or if the function code is cancel and the INTG had not been sent through the network, BQRD performs a fork/execute to call TLP with cancel JOBID.

After the TLP processing is completed, BQRD determines if there is a response processor to be called. If there is, GFUNLOCK is called to unlock GJOBS, BQCALLR is called to call the response processor, and GFLOCK is called to relock GJOBS. If a GJOBS item was deleted in the preceding processing, then BQRD unlinks any associated data file. BQRD writes the

current GJOBS item and determines if the function specifies an input to the network. If it does, then BQRD copies the input data file to its own data file (if applicable), and writes GRESP, if the job has a response processor. If the retransmit flag is set, BQRD searches GJOBS to see if the referenced INTG is still pending. If it is, then BQRD sets the function code to INTG, resets the retransmit flag, and goes to previously described processing for input to the network beginning with a check to see if either TAS or the receiving host is down.

BQRD is now ready for its cleanup processing. First GFUNLOCK is called to unlock GJOBS. If the function is either a READY or BREAK input to the network, then the host ID is set to TAS, a host status of up or down is saved, and BQRD goes to the previously described processing for a host status change. If the function is a RCPT for a SRVC (BREAK), then a host status of down is saved and BQRD also goes to the host status change processing described earlier. If the function is a logged out message and the logged out job was an automatic trace, then BQERROR is called to exit. If the function is a logged out message and the job is not an automatic trace, then the job type, the function code and the host ID are saved, BQCALLER is called to call SRESP, and BQERROR is called to exit.

If the lost INTG flag is set, then BQRD clears it, sets the function code to lost INTG, and goes back to the initial processing as described at the beginning of this section after the function code and host ID have been picked up.

If there is a network response file, then BQRD unlinks it. BQERROR is then called to exit. (See figure 16 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

BQPQP (Pending Query Processor) - BQPQP locks (via GFLOCK) the application's GJOBS file and opens GJOBS. BQPQP reads GJOBS and for

each job entry, BQPQP follows the subsequent procedure. A switch is made on the job's type of which there are six: INTG, SRVC(TRACE), SRVC(PRINT), SRVC(CHECK), SRVC(READY), and SRVC(BREAK). If the job is an INTG and has not been sent through the network, BQNET is called to send this INTG through the network. If a job is a SRVC(TRACE) but not an automatic trace, BQNET is called to send this job through the network. If the job is an automatic TRACE, a SRVC(READY), a SRVC(BREAK), or a SRVC(CHECK), the GJOBS entry for this job is set to zeroes and this job's BQRD data file (if it exists) is unlinked. A SRVC(PRINT) job is sent through the network via BQNET. For all cases, BQPQP next writes the GJOBS entry and checks to determine whether more jobs exist in GJOBS. If so, the preceding processing is repeated; otherwise, GJOBS is closed and unlocked (via GFUNLOCK), and BQPQP returns to the calling function.

BQTRACE (Send Automatic Traces) – This function initially reads the application's GJOBS file and searches for the next job. If the job is an INTG and if this INTG has not been sent through the network, function BQNET is called to send this INTG through the network and to update GJOBS. Once function BQNET has returned, BQTRACE writes this updated GJOBS item. If, on the other hand, the job is an INTG but the INTG has already been sent through the network, a new job has to be created in order to send a SRVC(TRACE) for that INTG through the network. First the next sequential JOBID is retrieved via GFJOBID. BQTRACE then finds the next empty item in GJOBS, and stores this SRVC(TRACE)'s job data in GJOBS. The BQRD data file is created and the traced INTG's JOBID is written into this file. BQNET is called to send this SRVC(TRACE) through the network and to update GJOBS. Next, BQTRACE writes this GJOBS item. Lastly, if the job is not an INTG, or if the INTG has been processed as above, BQTRACE continues its search through GJOBS. Once there are no more jobs in GJOBS, GJOBS is unlocked via GFUNLOCK and the function returns.

AD-A050 967

LOGICON INC SAN DIEGO CALIF

F/G 9/2

TERMINAL ACCESS SYSTEM (TAS) SYSTEM DESIGN SPECIFICATION.(U)

JAN 78 R M BARNHART, L R ERICKSON

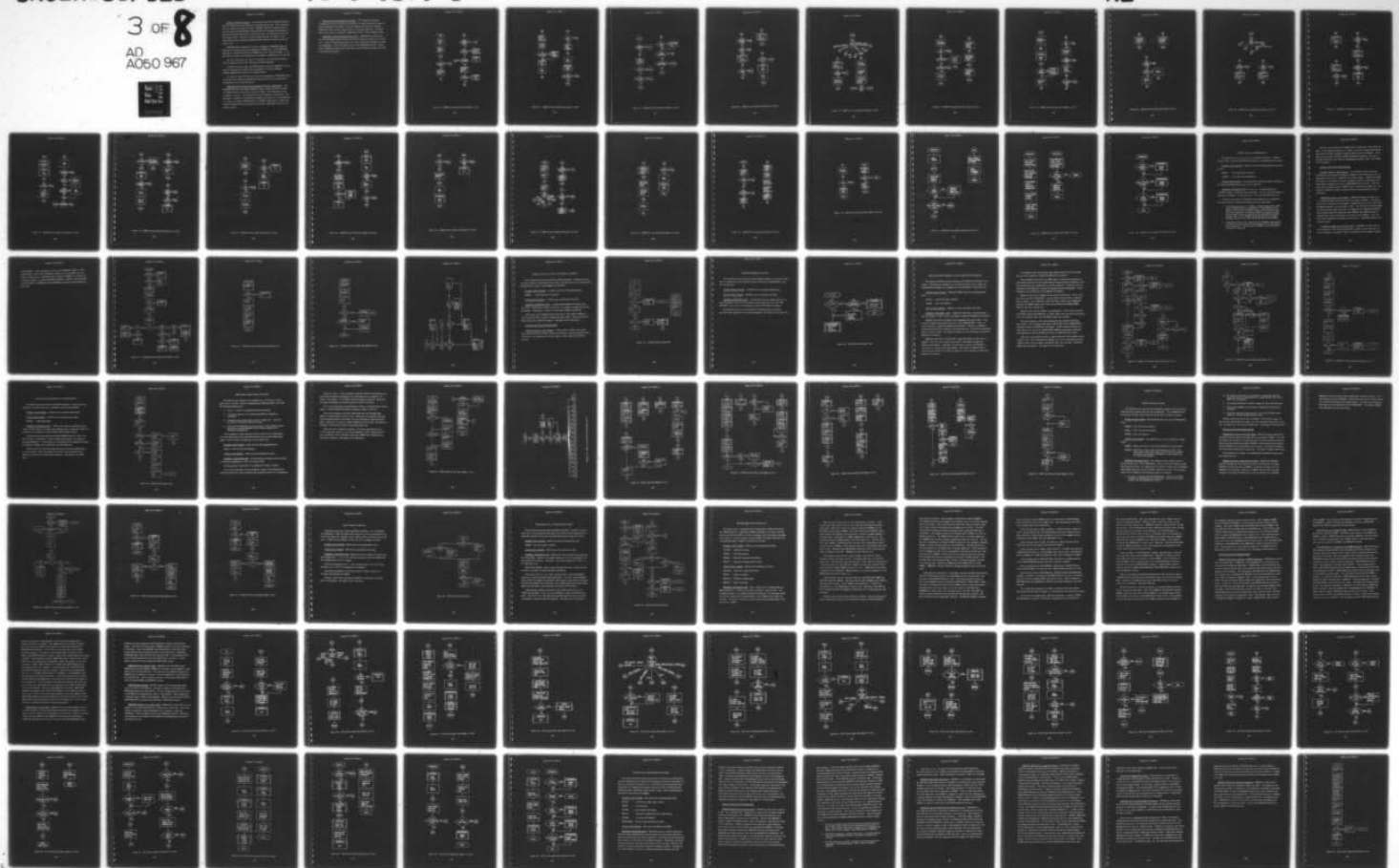
N00014-76-C-0899

UNCLASSIFIED

76-C-0899-6

NL

3 OF 8
AD
A050 967



BQNET (Call INI Process) – This function calls the Intelligent Network Interface (INI) Process to send the job through the network. INI is activated via a fork/execute sequence of calls. BQNET waits for the INI process to return a status on the attempted transmit before it continues processing. If the status indicates that the job could not be sent, the function returns with INI's status value. However, if the transmission was successful, a job transmitted status is stored in GJOBS and the function returns a successful status value.

BQSTRES (Store Response Processor Pathname)–BQSTRES opens the batch query response processor pathnames (GRES P) file. Then BQSTRES attempts to read the first block of GRESP into a 512 character buffer. If the block does not exist, each character in the buffer is initialized to a minus one.

Once the buffer has been read or initialized, it is determined whether there is enough room for the response processor pathname.

If sufficient room exists, the GRESP pointer where the pathname is to be stored is inserted into GJOBS and the pathname is stored in GRESP. Finally, BQSTRES returns to the calling function.

If, however, there did not exist room for the pathname, BQSTRES reads in the next block of data of GRESP and repeats the above processing until the pathname is successfully stored.

BQDLRES (Pick Up and Delete Response Processor Pathname) – The BQDLRES function first opens GRESP and then computes the block number for GRESP based on the pointer stored in GJOBS. The appropriate block of GRESP is read in. The response processor pathname is picked up. The pathname in GRESP is then deleted by setting each character of the pathname to a minus one and the GRESP pointer in GJOBS to set to zero. Finally the modified GRESP block is written out and closed. BQDLRES returns to the calling function.

BQCALLR (Call Response Processor) — The BQCALLR function initially performs the fork/execute sequence to call the response processor whose pathname is specified. Once the response processor is finished, BQCALLR checks to see whether there is a network input file. If the file does exist, the file is unlinked. BQCALLR returns to the calling function.

BQERROR (Output Message and/or Exit) — BQERROR is called with an error/message number or zero and an exit flag set to either zero or non-zero. If the error/message flag is set, then the specified message is output via GFERROR. If the exit flag is not set, then BQERROR returns. Otherwise, BQERROR unlocks any locked files via GFUNLOCK and then exits with the specified exit value.

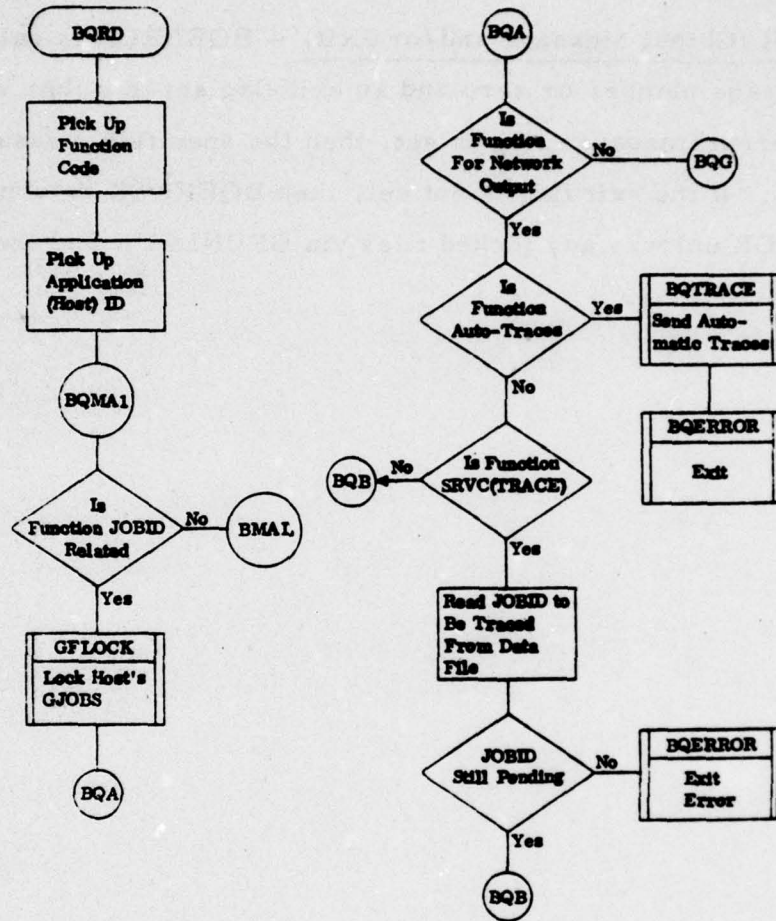


Figure 16. BQRD Process Data Flow (Sheet 1 of 22)

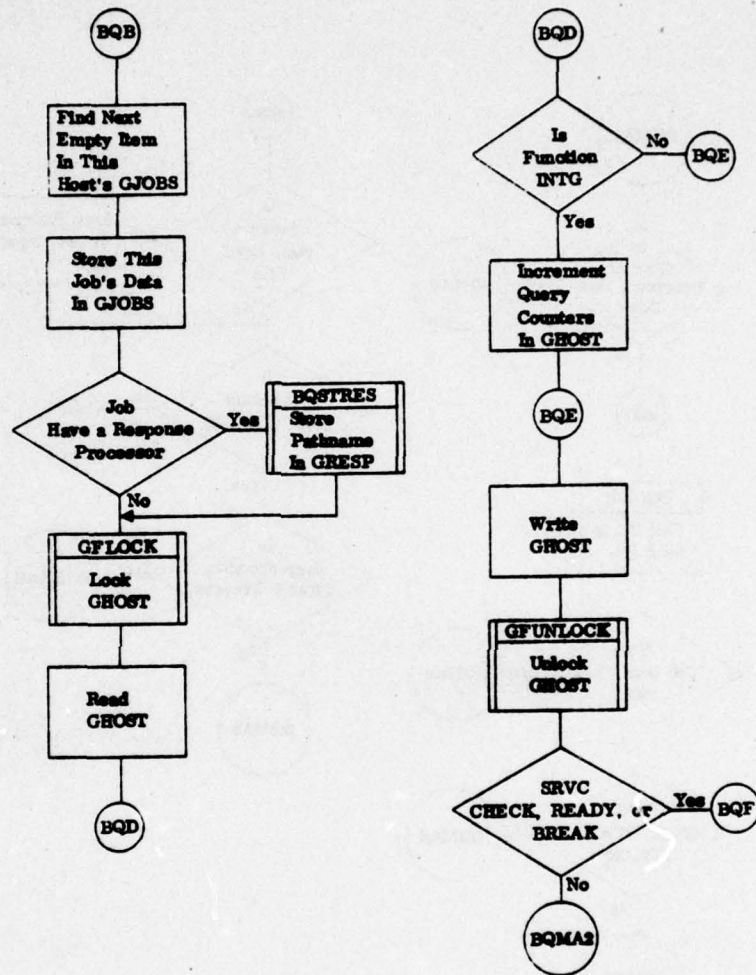


Figure 16. BQRD Process Data Flow (Sheet 2 of 22)

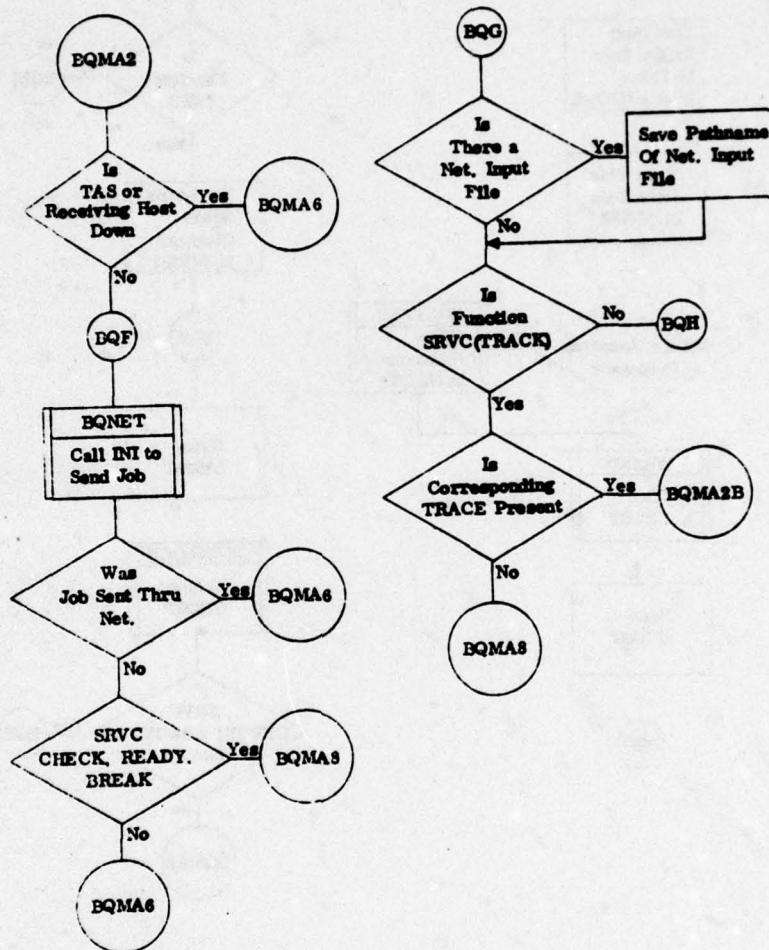


Figure 16. BQRD Process Data Flow (Sheet 3 of 22)

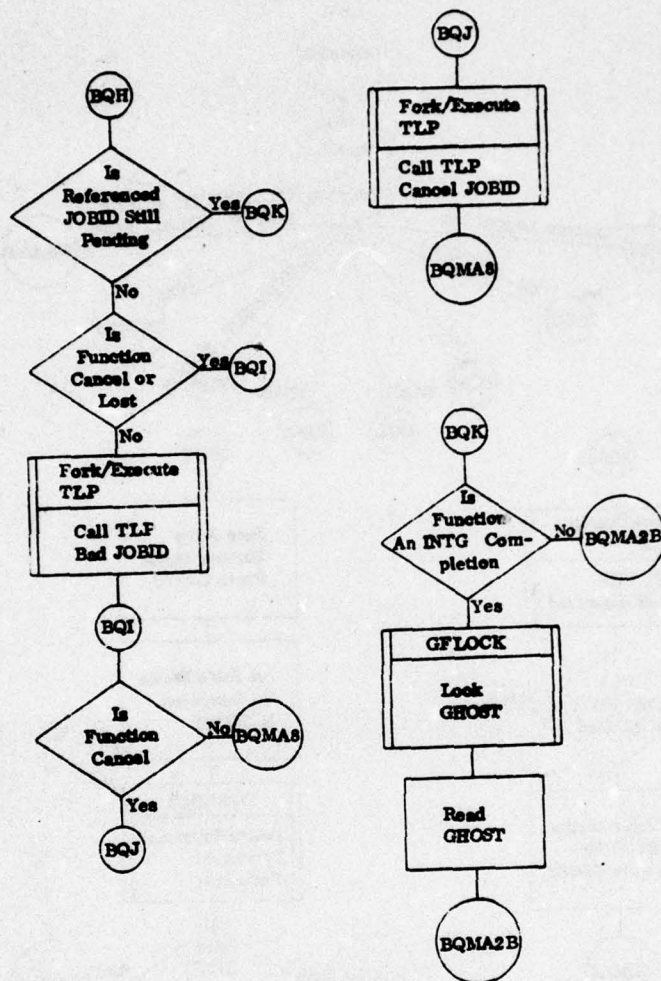


Figure 16. BQRD Process Data Flow (Sheet 4 of 22)

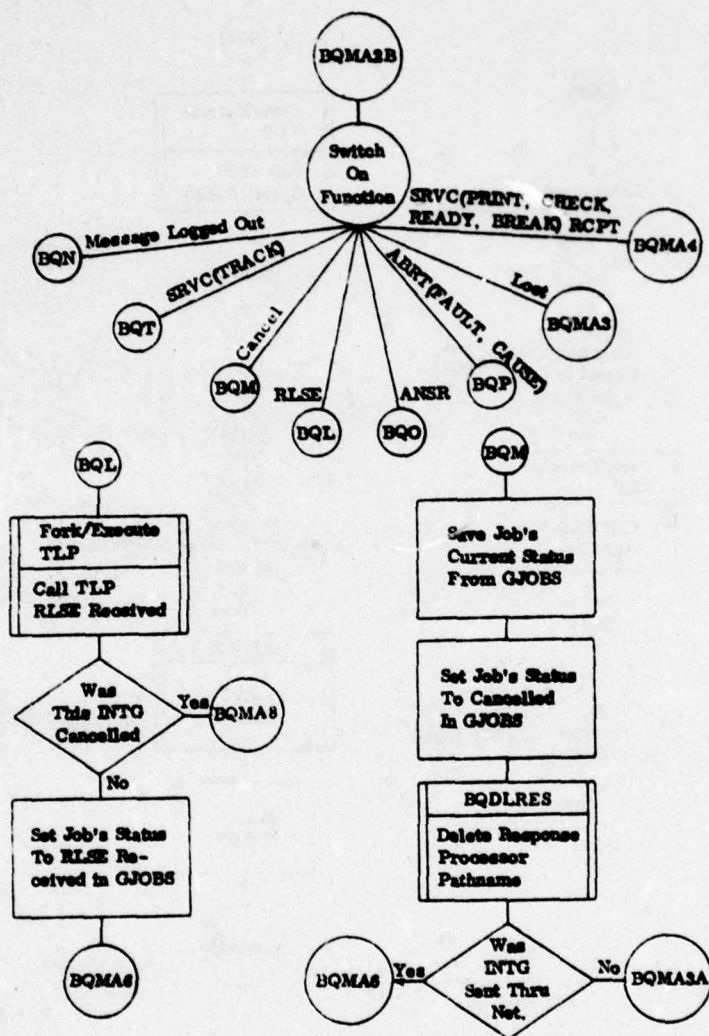


Figure 16. BQRD Process Data Flow (Sheet 5 of 22)

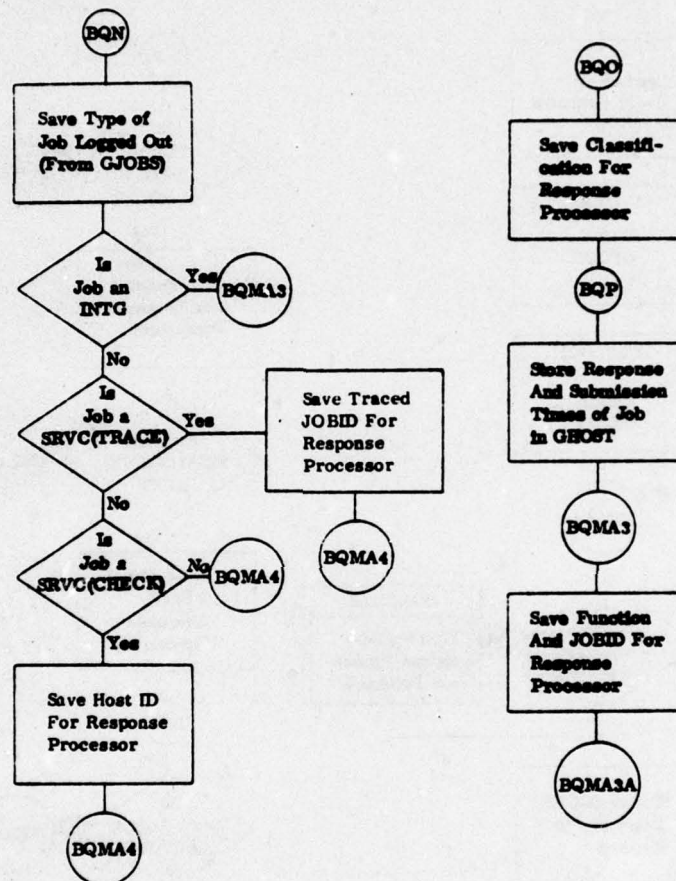


Figure 16. BQRD Process Data Flow (Sheet 6 of 22)

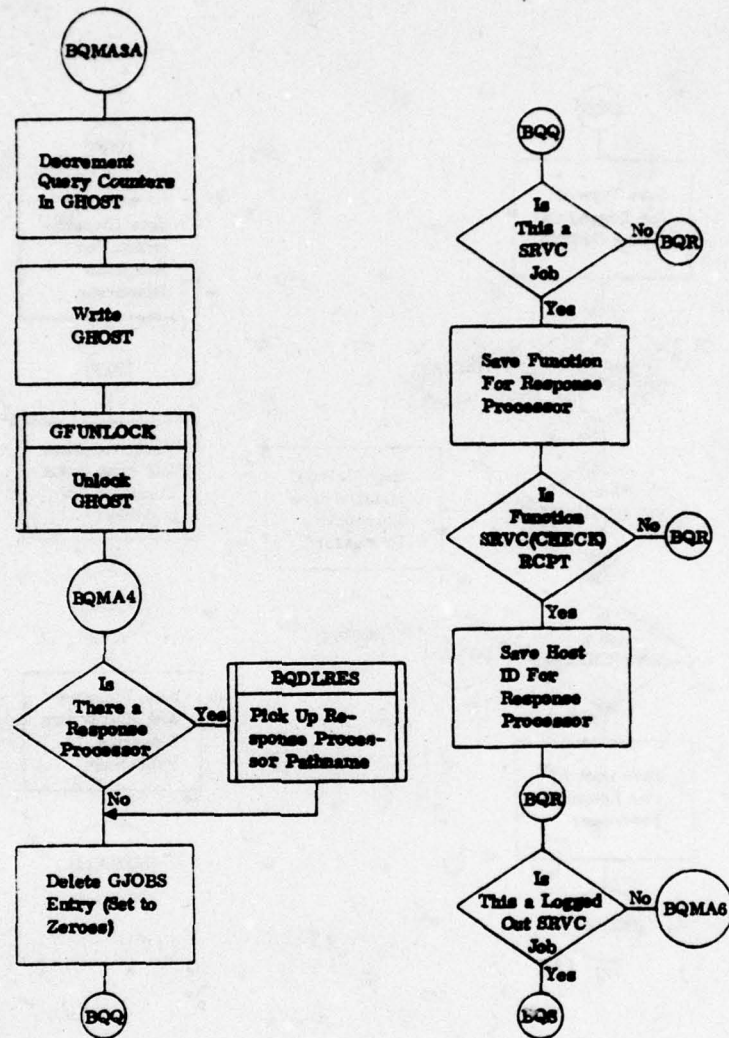


Figure 16. BQRD Process Data Flow (Sheet 7 of 22)

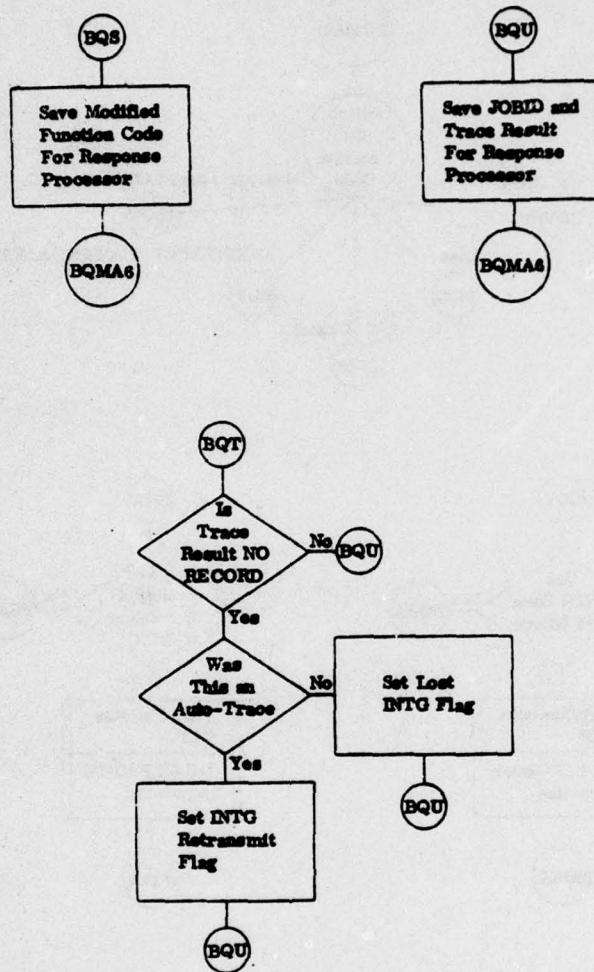


Figure 16. BQRD Process Data Flow (Sheet 8 of 22)

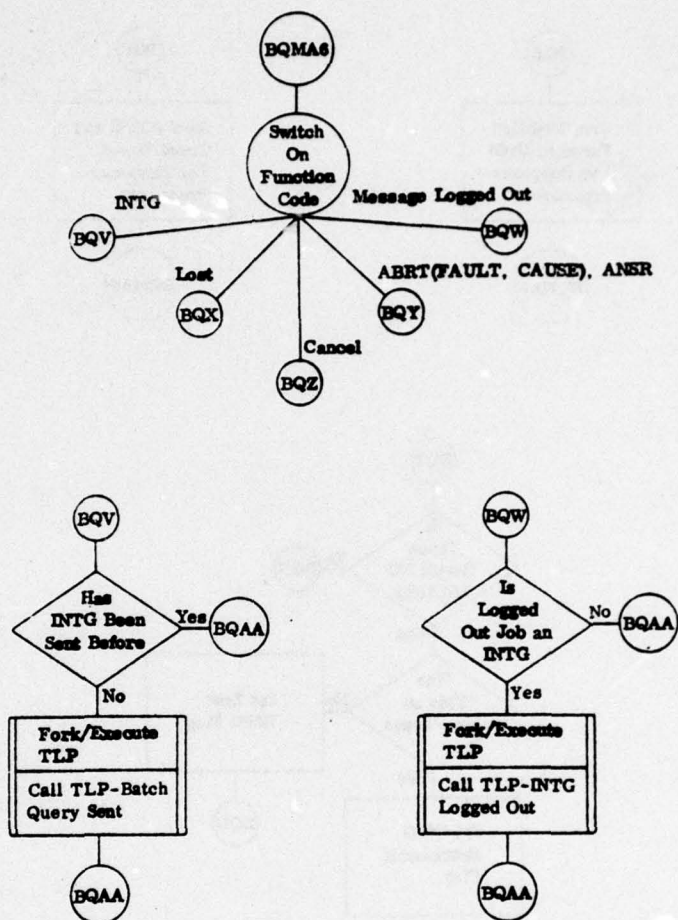


Figure 16. BQRD Process Data Flow (Sheet 9 of 22)

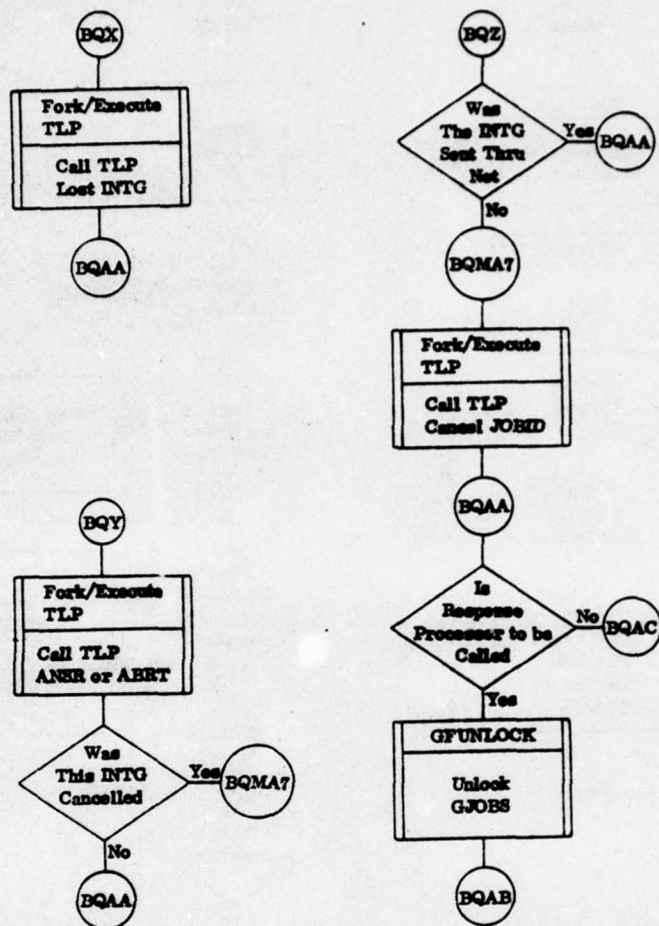


Figure 16. BQRD Process Data Flow (Sheet 10 of 22)

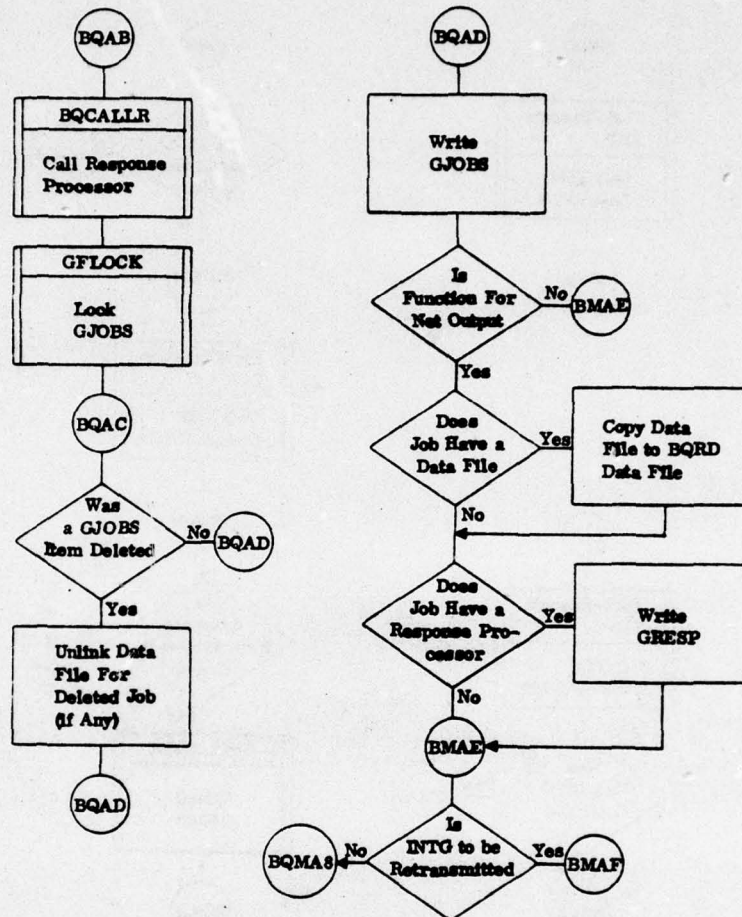


Figure 16. BQRD Process Data Flow (Sheet 11 of 22)

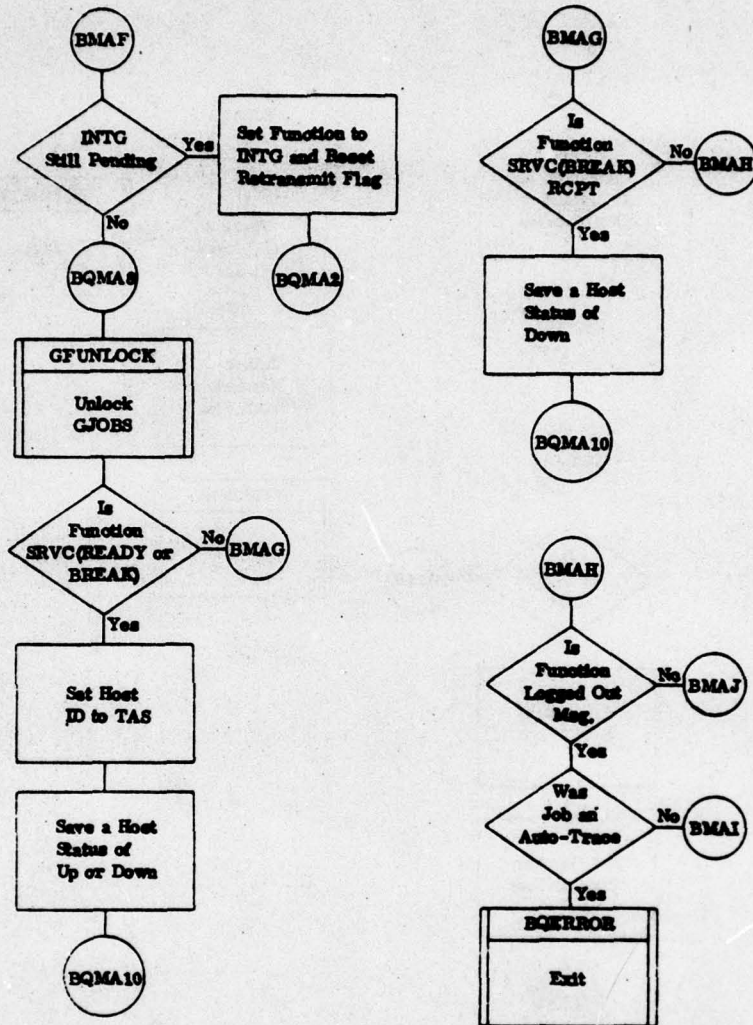


Figure 16. BQRD Process Data Flow (Sheet 12 of 22)

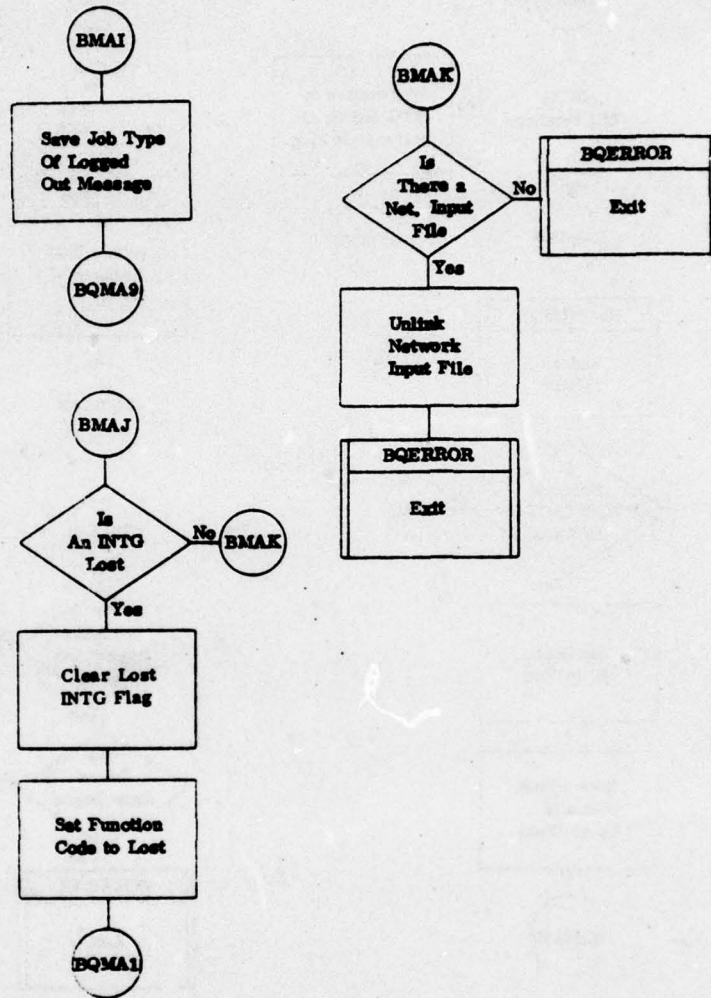


Figure 16. BQRD Process Data Flow (Sheet 13 of 22)

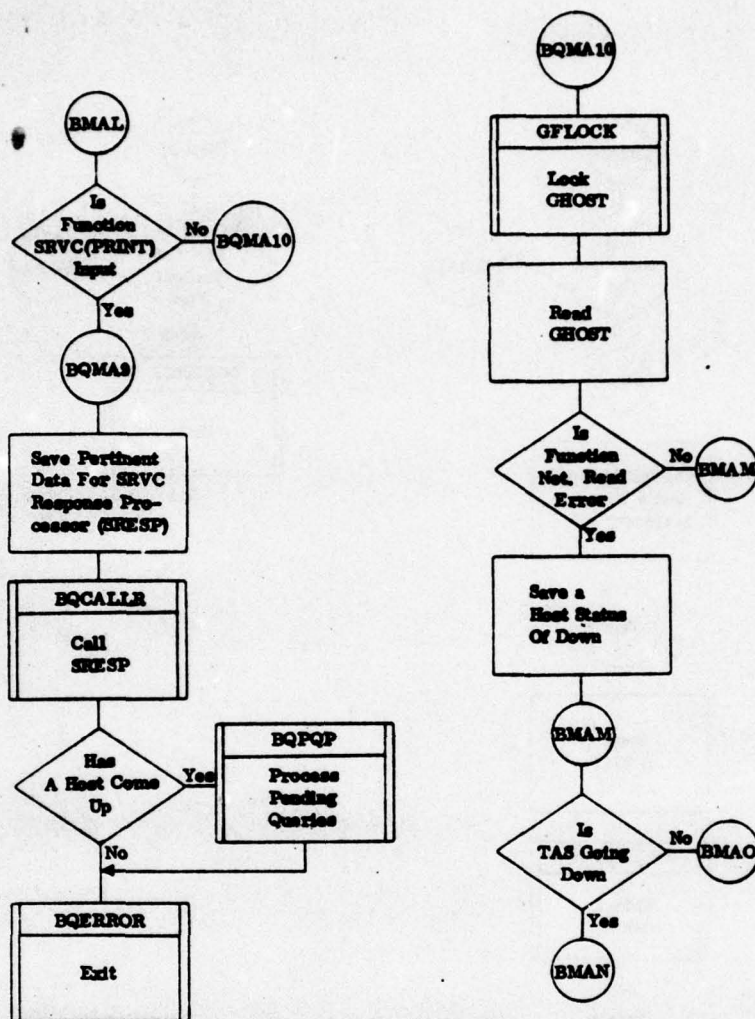


Figure 16. BQRD Process Data Flow (Sheet 14 of 22)

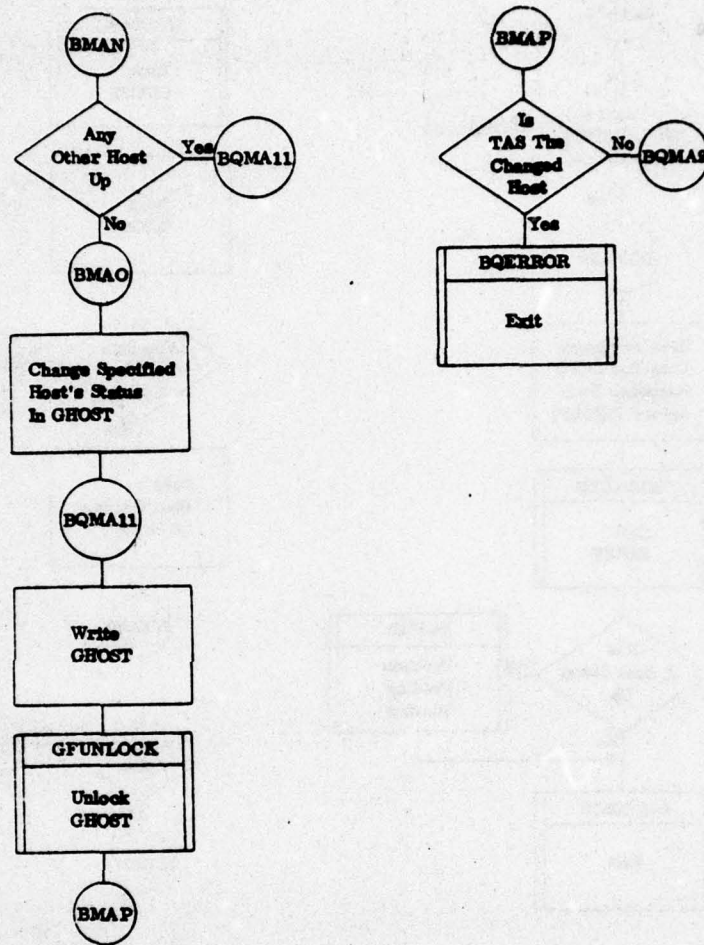


Figure 16. BQRD Process Data Flow (Sheet 15 of 22)

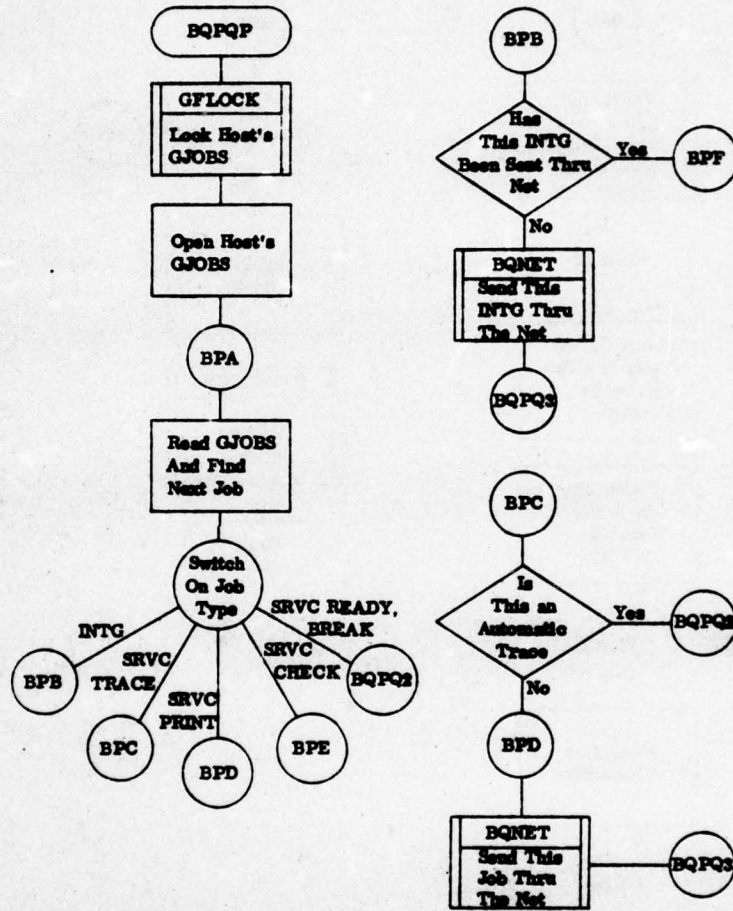


Figure 16. BQRD Process Data Flow (Sheet 16 of 22)

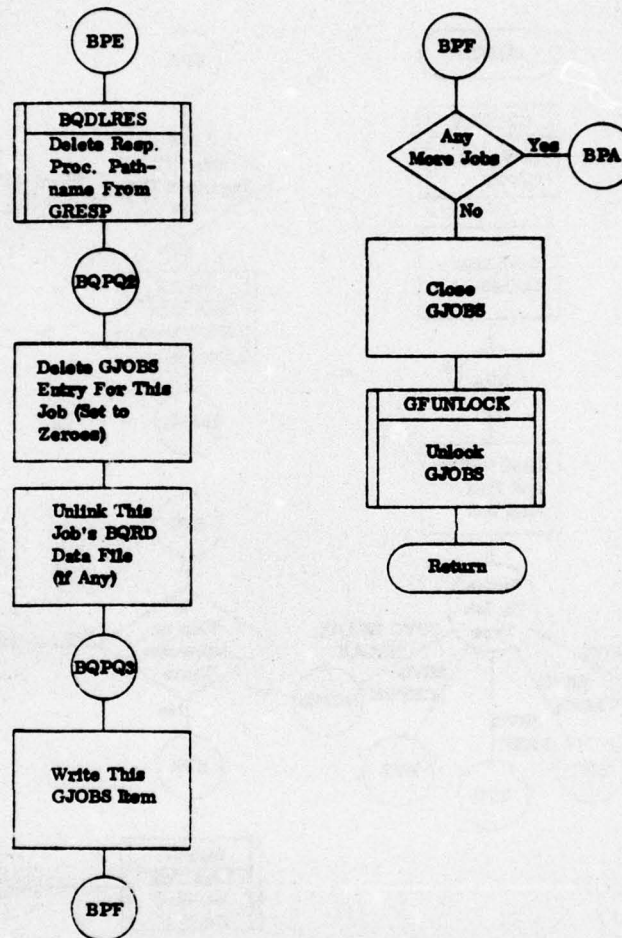


Figure 16. BQRD Process Data Flow (Sheet 17 of 22)

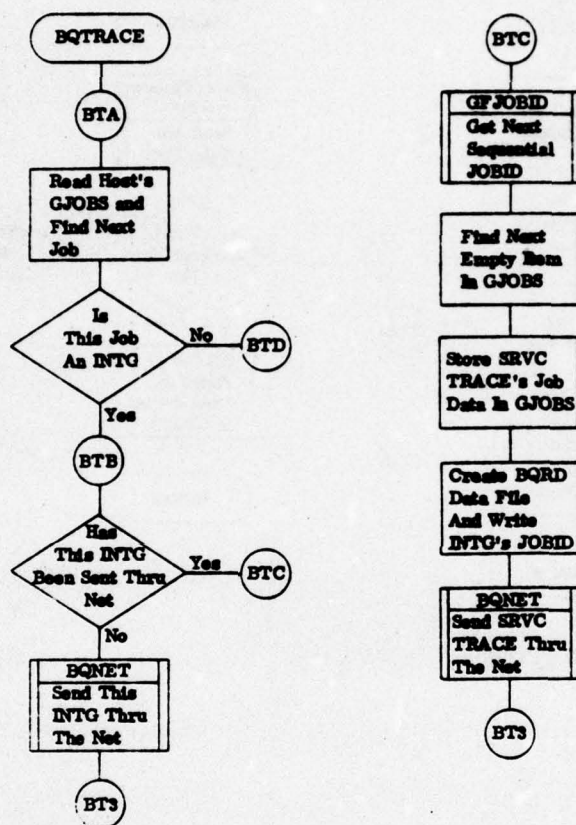


Figure 16. BQRD Process Data Flow (Sheet 18 of 22)

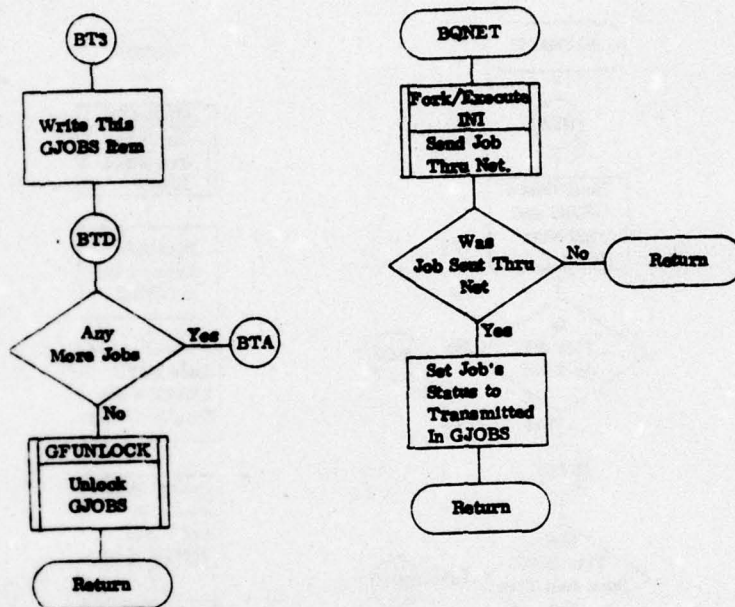


Figure 16. BQRD Process Data Flow (Sheet 19 of 22)

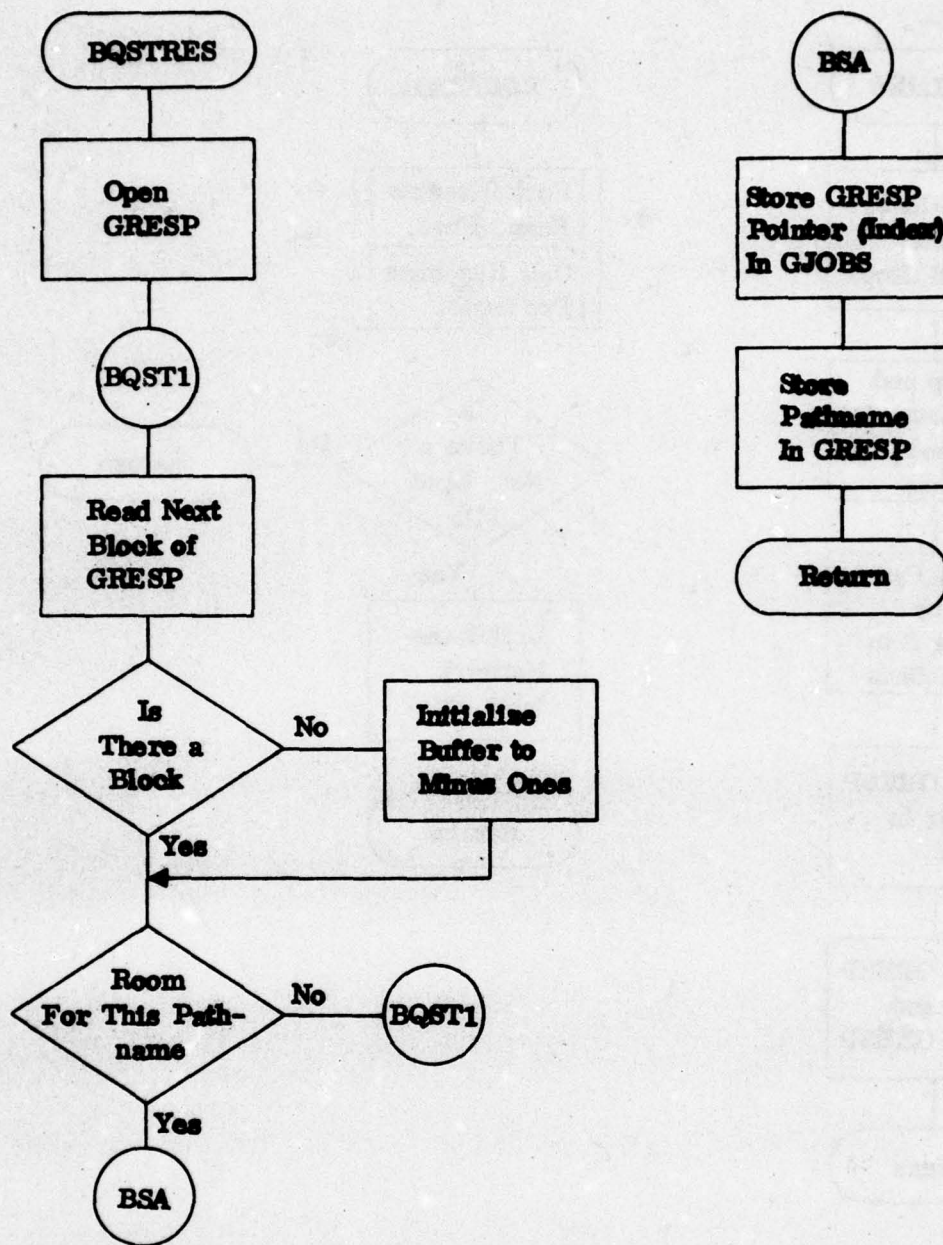


Figure 16. BQRD Process Data Flow (Sheet 20 of 22)

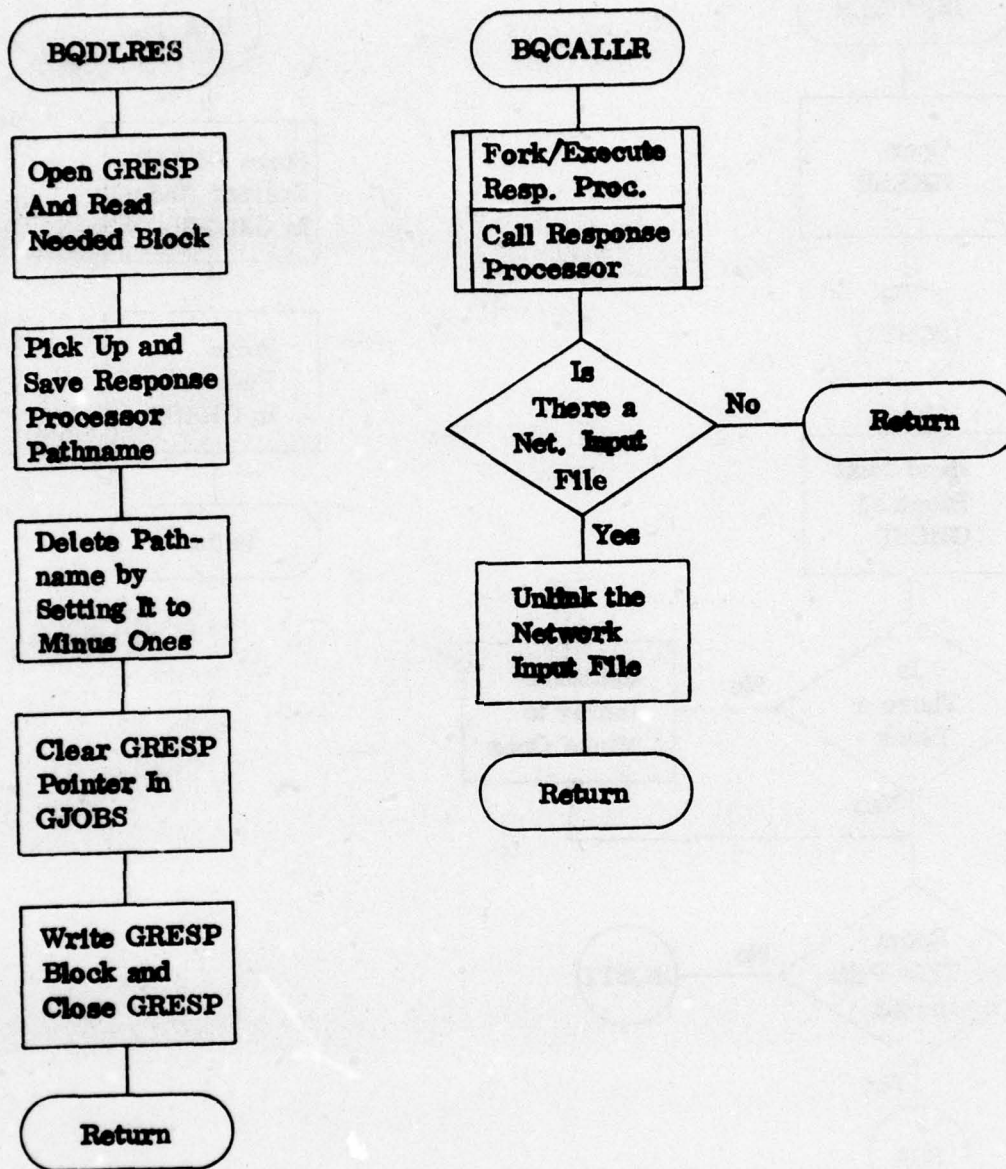


Figure 16. BQRD Process Data Flow (Sheet 21 of 22)

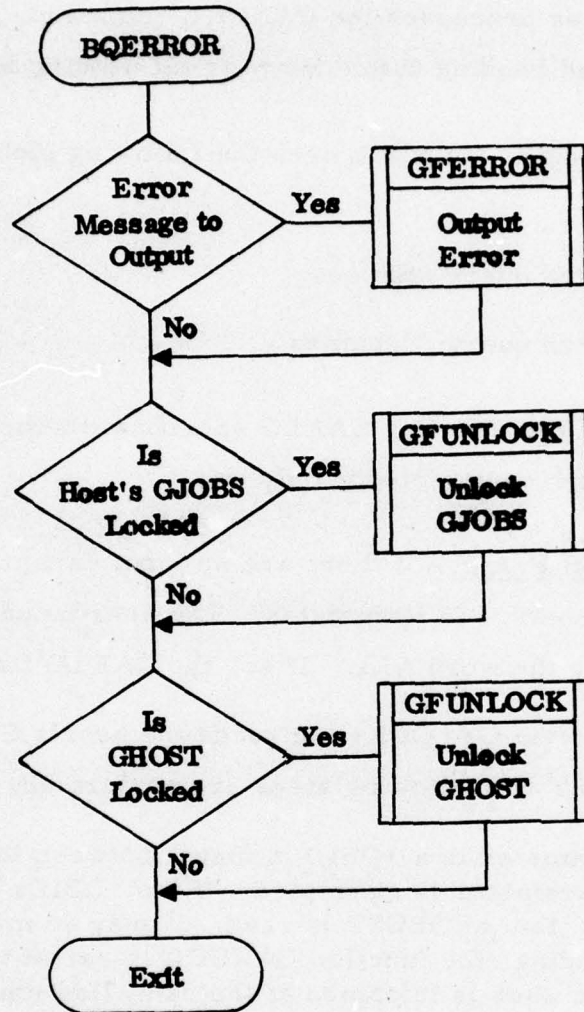


Figure 16. BQRD Process Data Flow (Sheet 22 of 22)

CANCEL (Cancel a Pending Query)

The CANCEL process processes the CANCEL command. CANCEL cancels either a specified pending batch query or all pending batch queries.

GLOBAL DATA USAGE - CANCEL uses the following global data structures:

GBQST - user batch query statuses

GBQFN - user batch query filenames

LOCAL DATA USAGE - The flag CAFLG specifies whether CANCEL is to cancel all pending batch query jobs or only one.

GENERAL PROCESS FLOW - If there are no input parameters, CANCEL outputs a diagnostic and terminates. The user input parameter is tested as to whether it is the word ALL. If so, the CAFLG flag is set.

CANCEL next locks (via GFLOCK) and reads the user's GBQST file. For each JOBID in GBQST the following steps are performed.

1. If the input parameter is a JOBID, a match between this JOBID and the input parameter is attempted. If the JOBIDs do not match, the next item of GBQST is read. If they do match and if the job is pending, the function CANCEL is called to cancel the job; then the user is informed of the cancelled job and the process exits. If the JOBIDs match but the job is no longer pending, the user is so informed and the process terminates.
2. If the flag CAFLG is set, and if the job is pending, the function CANCEL is called to cancel the job and the user is informed of the cancelled job. The next item in GBQST is read in and the procedure is repeated.

When the end-of-file for the GBQST file is encountered, two checks are made. If the input parameter was a JOBID, an error message informing the user that the JOBID is invalid is output and the process terminates. If any jobs were cancelled, CANCEL simply terminates; otherwise, the user is informed that none of his jobs were pending and CANCEL exits. (See figure 17 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

CANCEL (Cancel a Batch Query) – The CANCEL function calls the Batch Query and Response Dispatcher (BQRD) to cancel the query from the pending job queue for the applicable host target system. BQRD will change the status to "cancelled" so that the response will not be forwarded to the Batch Query Response Processor (RESPONSE) when received. Next, the query file is unlinked and the GBQST entry is cleared and written out. The function CADELFS is called to delete the files associated with the batch query job.

CADELFS (Delete a Job's Files) – CADELFS deletes the files associated with the batch query job as they are stored in GBQFN. The pointer to the files is picked up from GBQST. CADELFS reads the first file's data in GBQFN and picks up the number of files. For each file stored in GBQFN for this particular batch query job, an empty item is written into GBQFN. After having deleted the filenames from GBQFN, the GBQFN counter is adjusted only if the cleared items were at the end of GBQFN. If so, the first item of GBQFN is modified and written out. CADELFS returns to the calling function.

CAERROR (Output Error and/or Exit) – CAERROR is called with an error number or zero and an exit flag set to either zero or non-zero. If an error number is specified, then the specified error message is output

via GFERROR. If the exit flag is not set, then CAERROR returns. If the GBQST file is locked, then CAERROR checks to see if any jobs were cancelled. If there were, CAERROR either writes out GBQST or, if there are no jobs remaining, recreates GBQST and GBQFN. GBQST is then unlocked via GFUNLOCK. CAERROR calls GFERROR to output an end message and then exits.

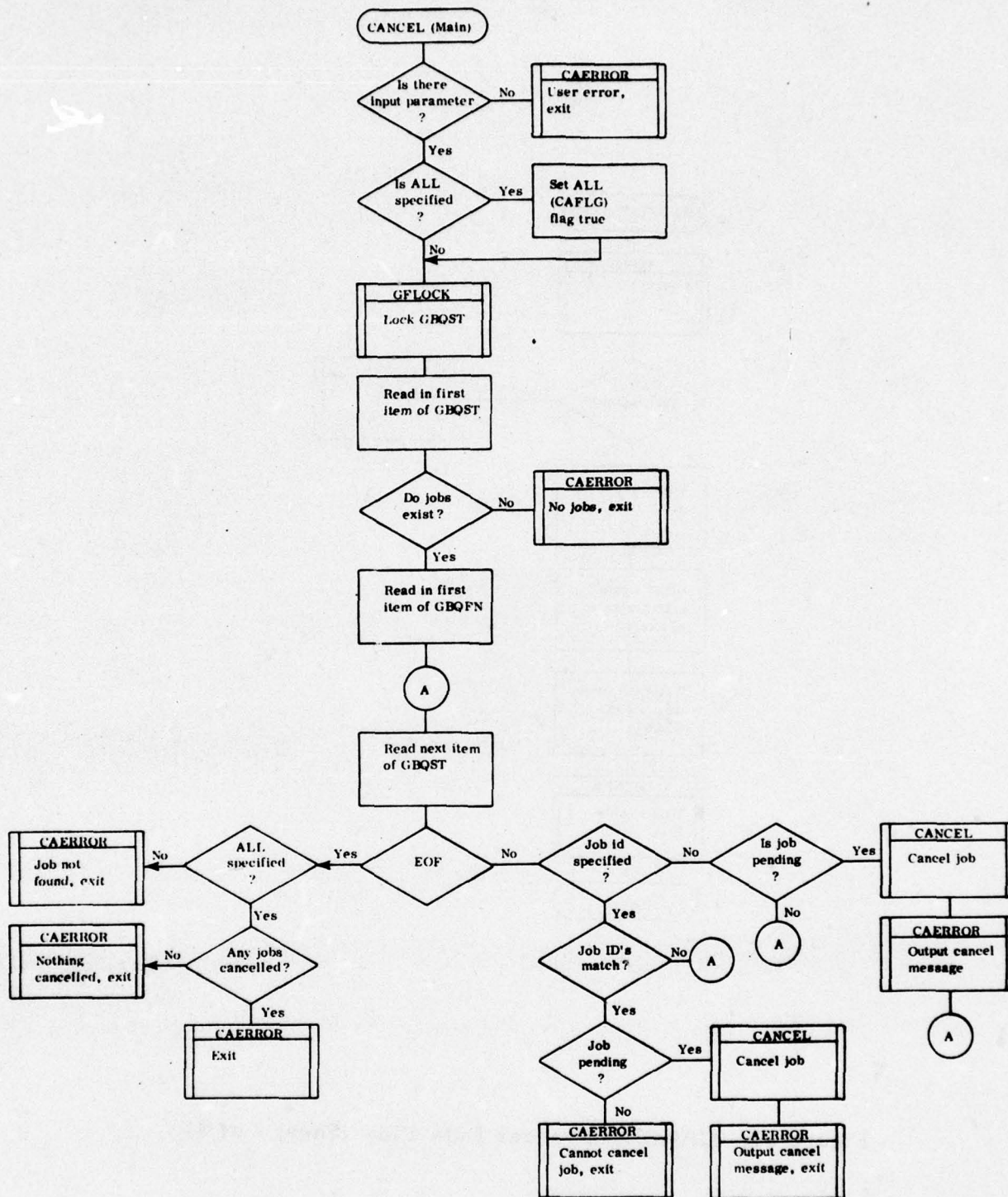


Figure 17. CANCEL Process Data Flow (Sheet 1 of 4)

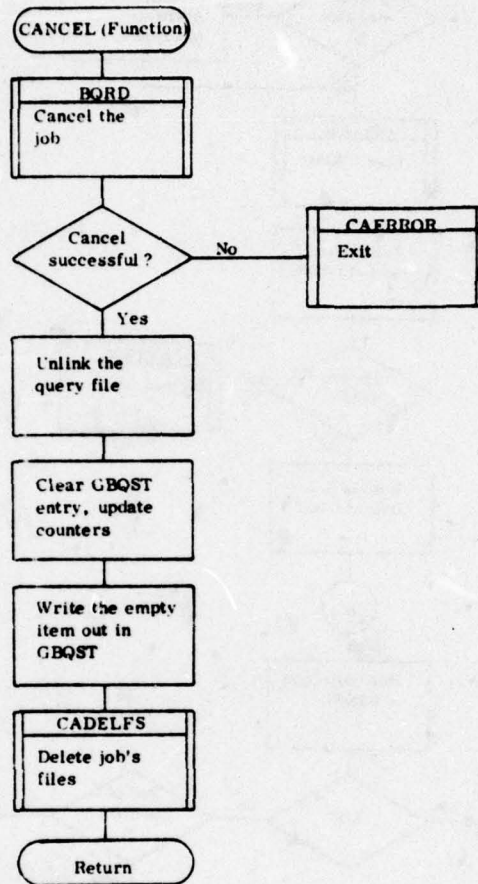


Figure 17. CANCEL Process Data Flow (Sheet 2 of 4)

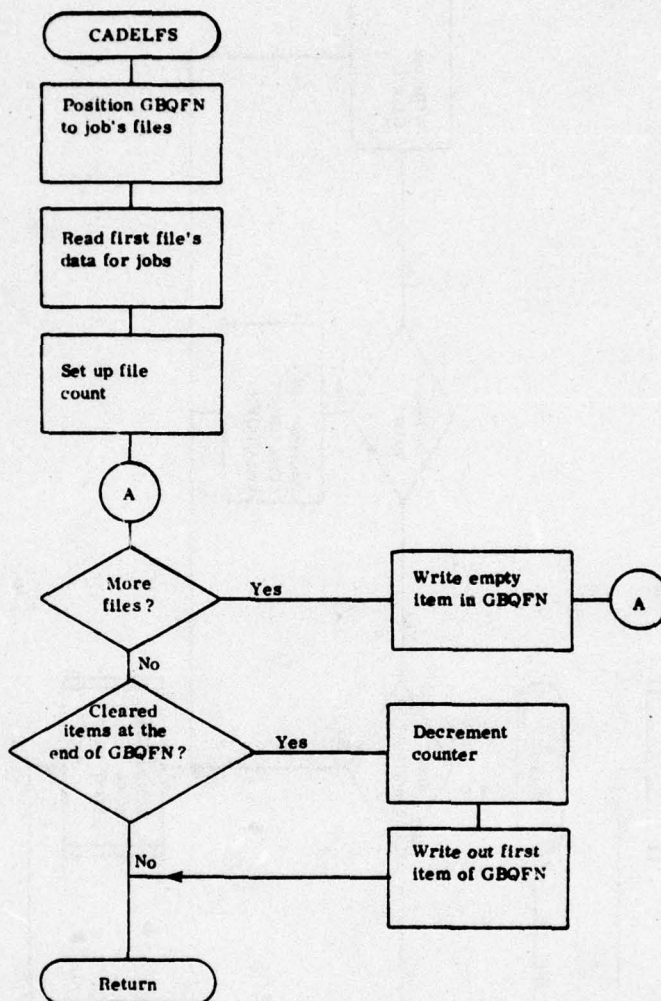


Figure 17. CANCEL Process Data Flow (Sheet 3 of 4)

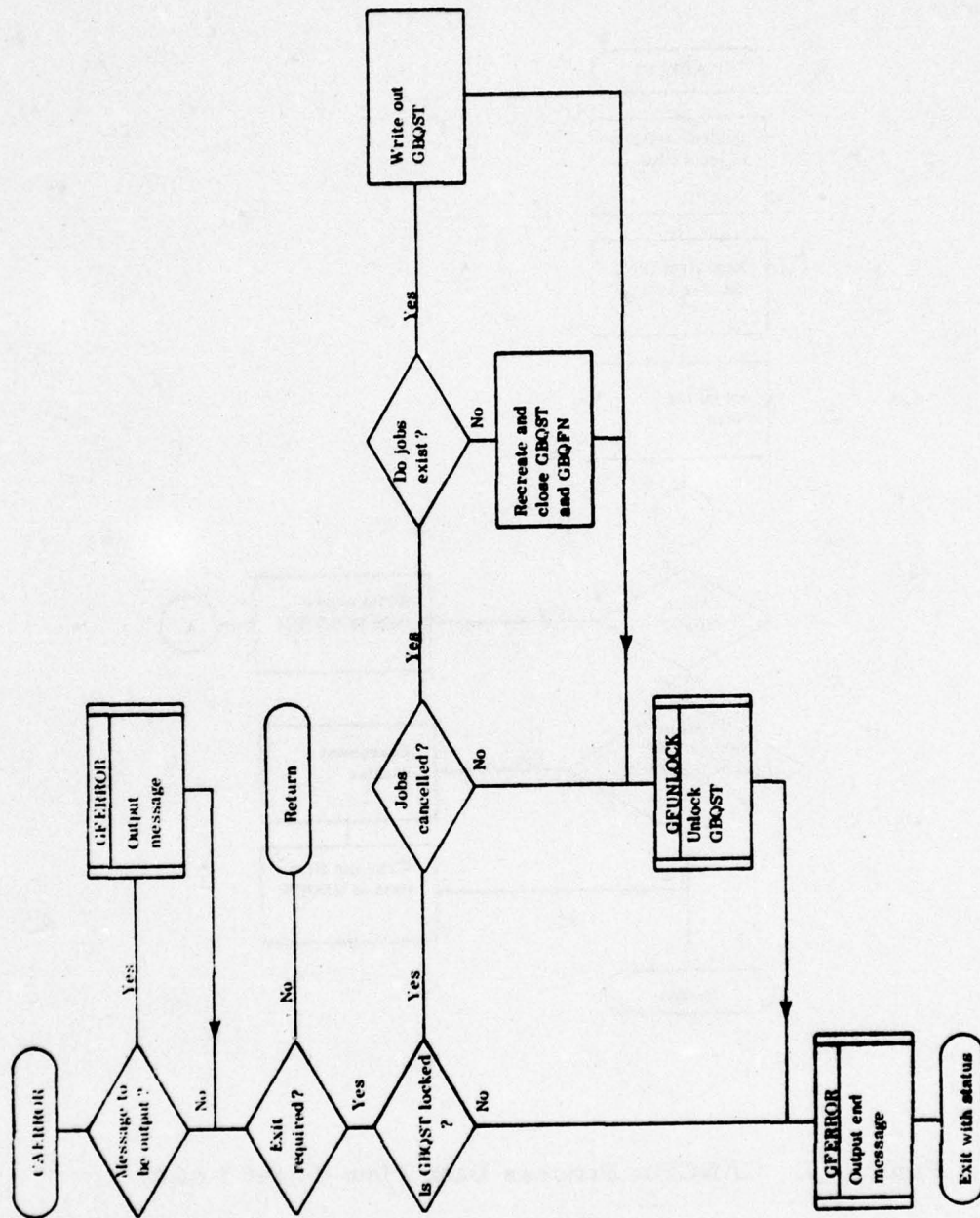


Figure 17. CANCEL Process Data Flow (Sheet 4 of 4)

CLOSE (List User's Jobs to be Purged on LOGOFF)

The CLOSE process processes the CLOSE command. It allows a user to obtain the status of all his batch query jobs, with those jobs to be automatically purged when he logs off TAS flagged with "#####".

GLOBAL DATA USAGE - CLOSE uses the following global data:

GBQST - user batch query statuses

LOCAL DATA USAGE - CLOSE uses no pertinent local data.

GENERAL PROCESS FLOW - CLOSE opens GBQST and reads in the first item. If there are no jobs for the user, a message is output and CLOSE terminates. Otherwise, a header for the output listing is generated.

Each user's job in GBQST is then processed. Each item of GBQST is read in and CLOUST is called to output that job's status. When all jobs have been processed, CLOSE terminates. (See figure 18 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

CLOUST (Output a Job's Status) - When called, CLOUST sets up the delivered/not delivered status, held/not held status, and the purgeable status of the job. The summary of the job's status is then output and CLOUST returns.

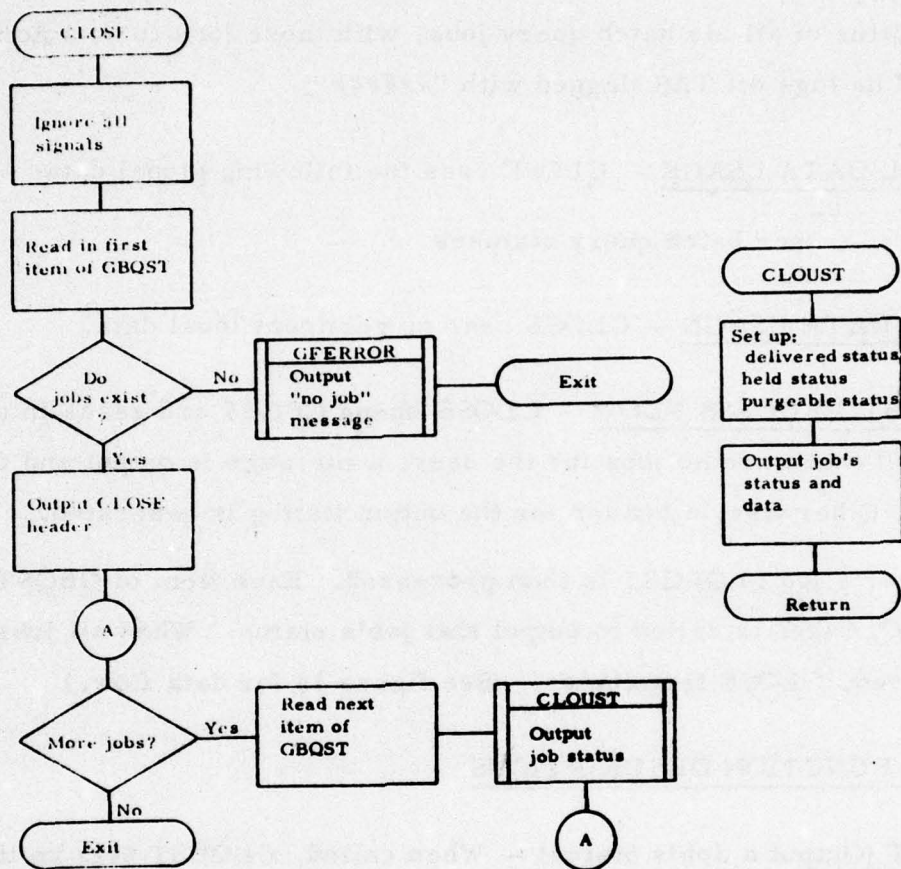


Figure 18. CLOSE Process Data Flow

DATEIS (Display/Set the Date)

The DATEIS process processes the DATEIS command. It allows all TAS users to obtain the current TAS time and/or allows the TASMASTER to set the current time.

GLOBAL DATA USAGE - DATEIS uses no pertinent global data.

LOCAL DATA USAGE - DATEIS uses no pertinent local data.

GENERAL PROCESS FLOW - DATEIS first checks whether there is an input argument. If so, a test is made to ensure that the user is the TAS-MASTER. If not, an error message is output and DATEIS terminates.

DATEIS then executes the UNIX DATE command process with the specified input argument (if any) and terminates. (See figure 19 for data flow.)

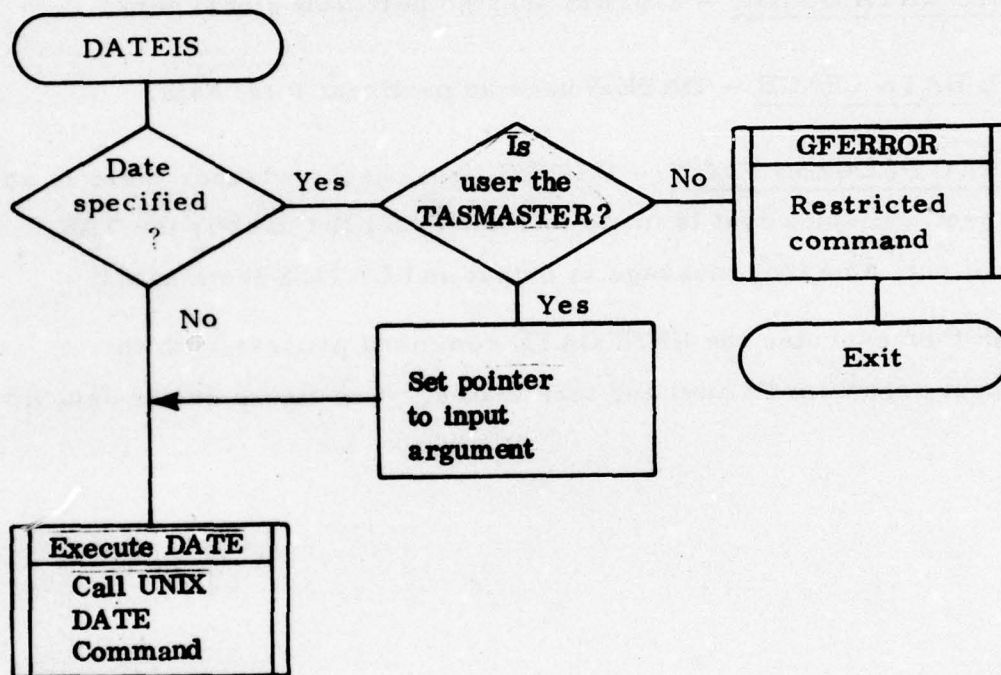


Figure 19. DATEIS Process Data Flow

DISPLAY/PRINT (Display or Print a Batch Query Response)

The DISPLAY/PRINT process processes the DISPLAY and PRINT commands. It displays the response to a user's batch query job or allows the TASMMASTER to display the response to a job belonging to a specified user.

GLOBAL DATA USAGE - DISPLAY/PRINT uses the following global data:

GBQST - user batch query statuses

GUDES - user descriptions

LOCAL DATA USAGE - DISPLAY uses no pertinent local data.

GENERAL PROCESS FLOW - DISPLAY determines if any arguments have been input. If not, an error message is output and DISPLAY terminates.

DISPLAY then checks whether three arguments have been input. If so, a check is made to determine if the user is the TASMMASTER. If not, an error message is output and DISPLAY terminates. Otherwise, GUDES is opened and searched for a match to the input user sni. The search terminates when a match is found. If no match is found, an error message is output and DISPLAY terminates.

DISPLAY then locks (via GFLOCK), opens and reads the first item of GBQST. If the user has no batch query jobs, a message is output and DISPLAY terminates. Otherwise, the job count is saved, and GBQST is searched for a match to the input JOBID. The search terminates when a match has been found; if no match is found, an error message is output and DISPLAY terminates.

If the GBQST item matching the input JOBID shows the job to be pending, an error message is output and DISPLAY terminates.

If the response to the job is an ANSR, then a fork/execute sequence is performed to call the Access Authorization Process (AAP) to determine if the user and terminal are authorized to view the response. If the user was not authorized, then DISPLAY deletes the specified job and terminates. If the terminal is not authorized, then DISPLAY terminates.

DISPLAY then checks whether the job has been delivered. If not, and if the user is not the TASMASTER, a test is made to determine if a response file is present. If so, DISPLAY calls TLP with a "batch query displayed" function, the GBQST status of the job is set to delivered and the GBQST item for this job is written out.

DISPLAY then unlocks GBQST via GFUNLOCK. If the actual command was PRINT rather than DISPLAY, a "print" flag is set, the user is requested to hit the "print-on-line" key and PRINT waits four seconds.

The query header is then output followed by the content of the batch query itself. If no response file exists, and the user is the TASMASTER, a message is output and DISPLAY terminates. If the response file does not exist and the user is not the TASMASTER, DISPLAY performs a fork/execute sequence to call the JOBSTAT process to output the job's status.

Otherwise, the response header is output followed by the contents of the response file. If the command was PRINT, the user is requested to hit the "print-on-line" key once again, and PRINT waits four seconds. DISPLAY/PRINT then terminates. (See figure 20 for data flow.)

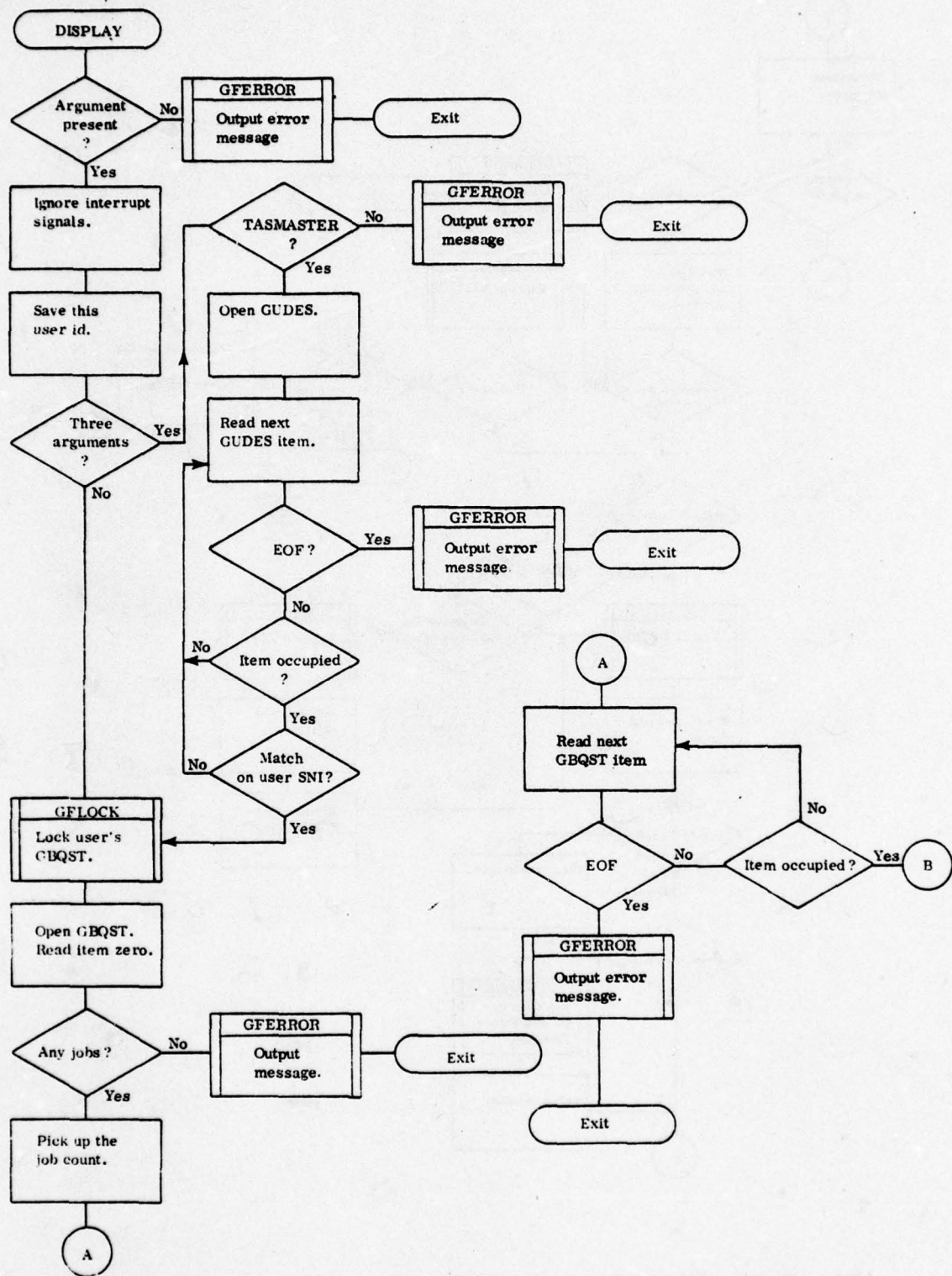


Figure 20. DISPLAY Process Data Flow (Sheet 1 of 3)

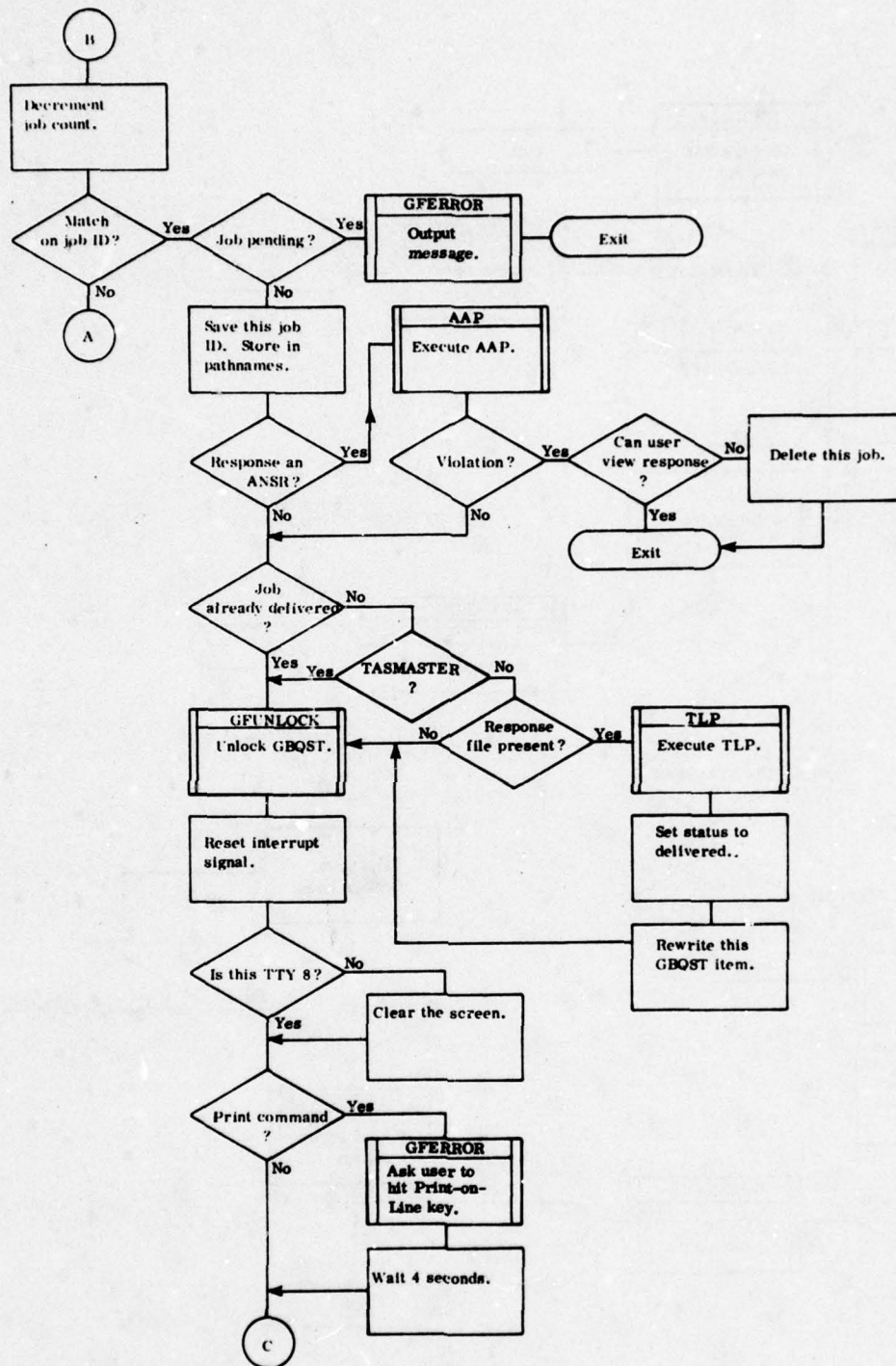


Figure 20. DISPLAY Process Data Flow (Sheet 2 of 3)

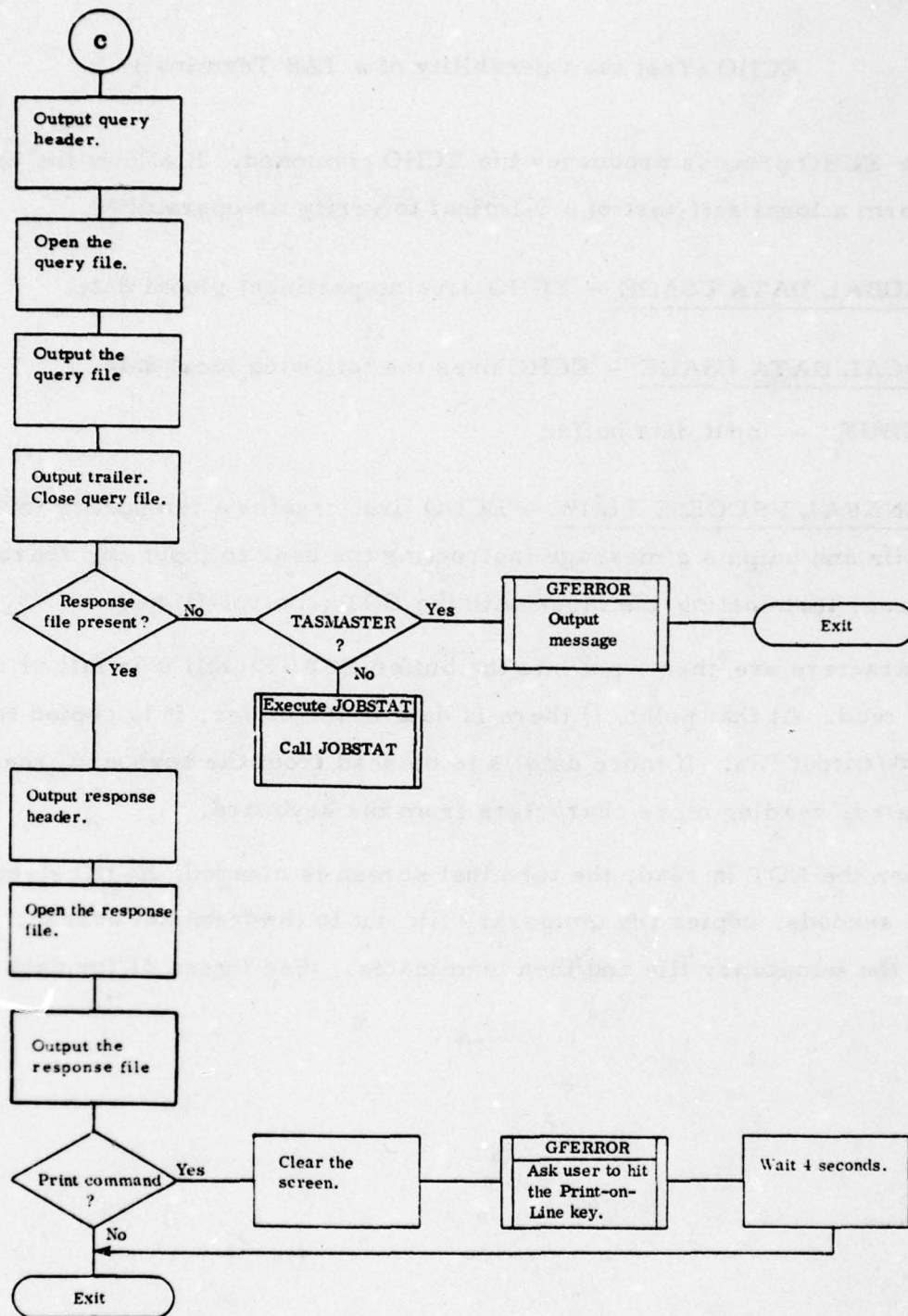


Figure 20. DISPLAY Process Data Flow (Sheet 3 of 3)

ECHO (Test the Operability of a TAS Terminal)

The ECHO process processes the ECHO command. It allows the user to perform a local self-test of a terminal to verify its operability.

GLOBAL DATA USAGE - ECHO uses no pertinent global data.

LOCAL DATA USAGE - ECHO uses the following local data:

ECBUF - input data buffer

GENERAL PROCESS FLOW - ECHO first creates a temporary input/output file and outputs a message instructing the user to input any characters he desires, terminating the block with the EOT (control-D) key.

Characters are then input into the buffer ECBUF until it is full or an EOT is read. At that point, if there is data in the buffer, it is copied to the input/output file. If more data is to be read from the keyboard, the loop is repeated, reading more characters from the keyboard.

When the EOT is read, the terminal screen is cleared, ECHO sleeps for five seconds, copies the temporary file out to the terminal screen, unlinks the temporary file and then terminates. (See figure 21 for data flow.)

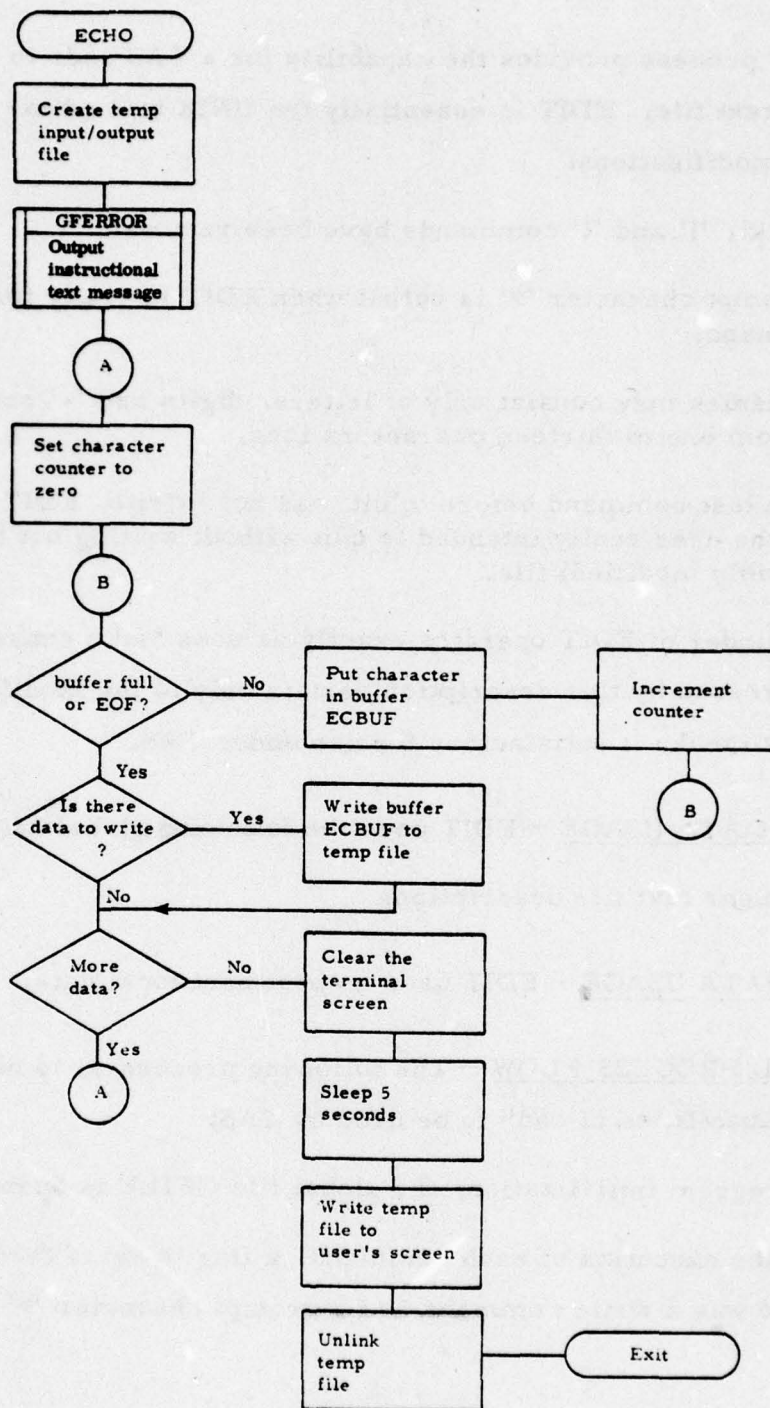


Figure 21. ECHO Process Data Flow

EDIT (Create and/or Edit a Text File)

The EDIT process provides the capability for a TAS user to create and/or edit a text file. EDIT is essentially the UNIX text editor "ed" with the following modifications:

1. The 'k', 'l' and '!' commands have been removed.
2. A prompt character '*' is output when EDIT is ready for a command.
3. Filenames may consist only of letters, digits and '.' and may be from one to thirteen characters long.
4. If the last command before 'q'uit was not 'w'rite, EDIT verifies that the user really intended to quit without writing out his (possibly modified) file.

The remainder of EDIT operates exactly as does "ed"; consequently, the areas addressed by this description pertain only to the modifications made to "ed" to make it satisfactory for use under TAS.

GLOBAL DATA USAGE - EDIT uses the following global data:

GFILE - user text file descriptions

LOCAL DATA USAGE - EDIT uses no pertinent local data.

GENERAL PROCESS FLOW - The following processing is done by EDIT to allow the capabilities of "ed" to be used by TAS:

During program initialization, the global file GFILE is opened.

Prior to the execution of each command, a flag is set if the last command executed was a write command and a prompt character '*' is displayed.

When an 'e'dit or 'r'ead command is entered, GFILE is searched for an entry with a filename matching the one referenced in the command. If a match is not found, a '?' is output and the command is ignored. Otherwise, the index into GFILE for the file is saved for later references.

If a 'w'rite command is issued and the current or referenced file already exists, a "REPLACING OLD FILE" message is output to the user.

If the user attempts to write to a file which does not already exist, GFILE is locked and item zero is read. If the user currently has the maximum allowable number of text files, an error message is output and EDIT exits. Otherwise, an entry is made in GFILE for the new file, and GFILE is unlocked. The new file is then created and written out.

When the 'q'uit command is entered, if the last command was not a 'w'rite command, and there is a current filename, EDIT requests that the user confirm his 'q'uit command. If he does, EDIT exits; otherwise the command is ignored. (See figure 22 for data flow.)

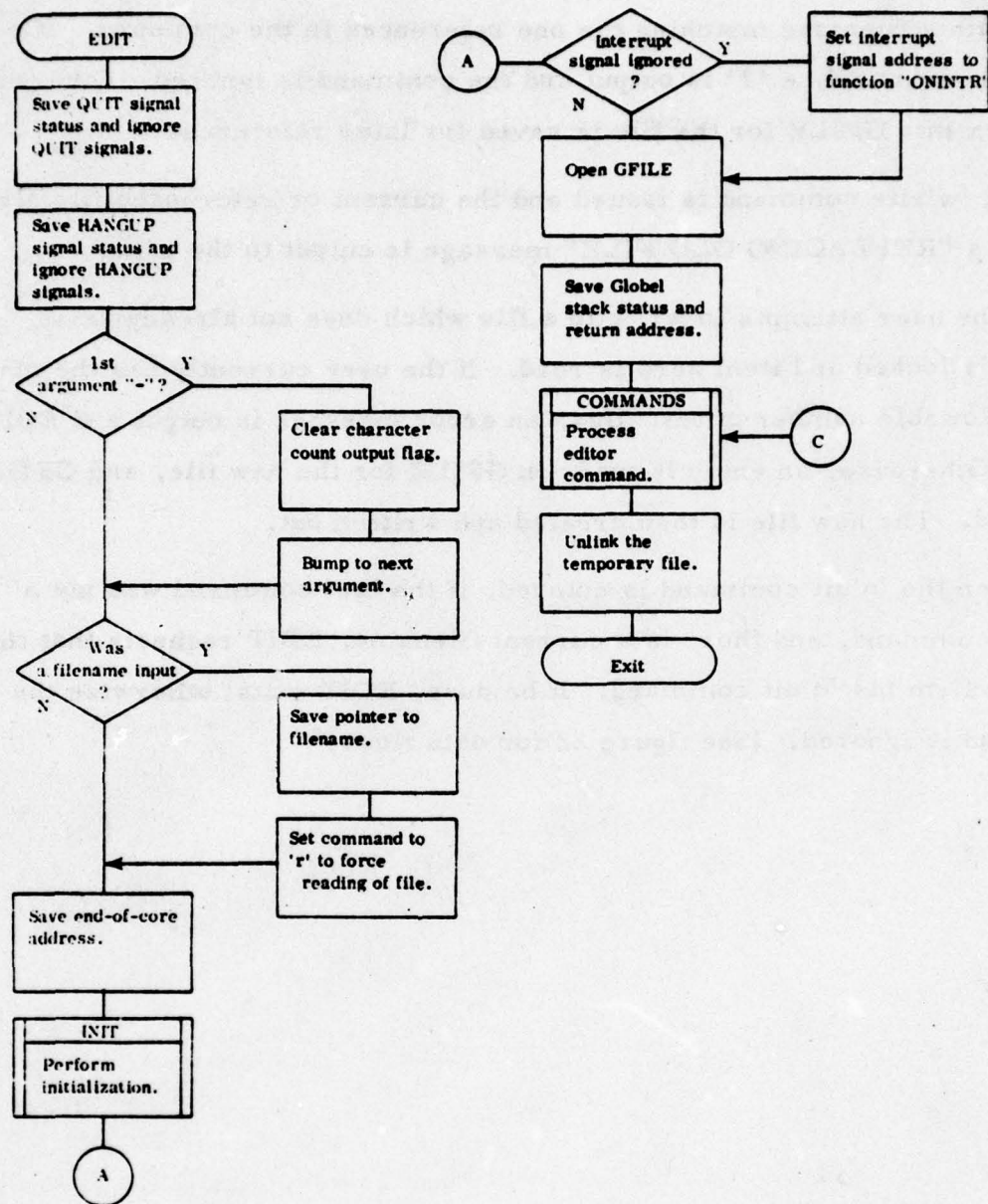


Figure 22. EDIT Process Data Flow (Sheet 1 of 7)

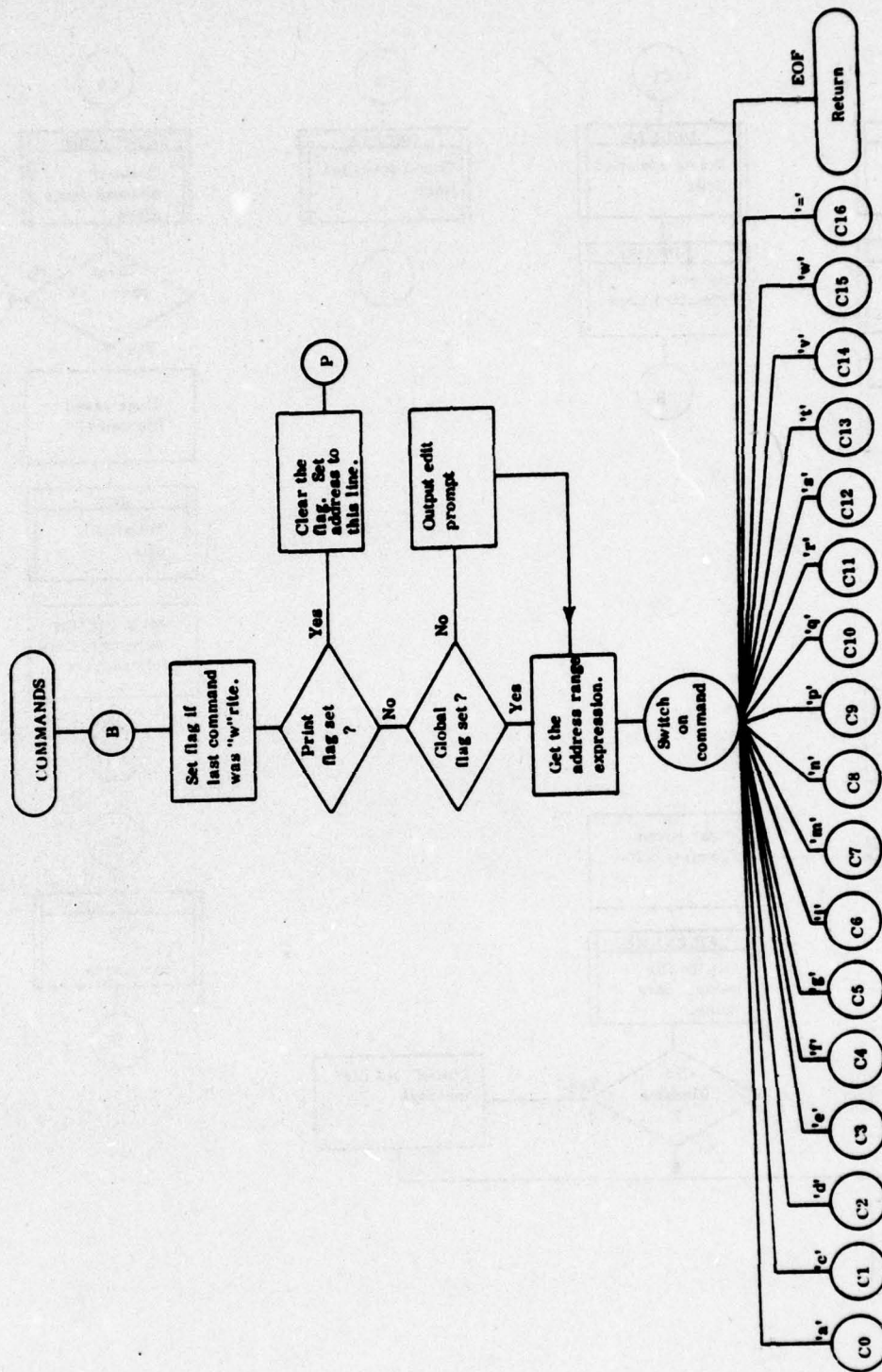


Figure 22. EDIT Process Data Flow (Sheet 2 of 7)

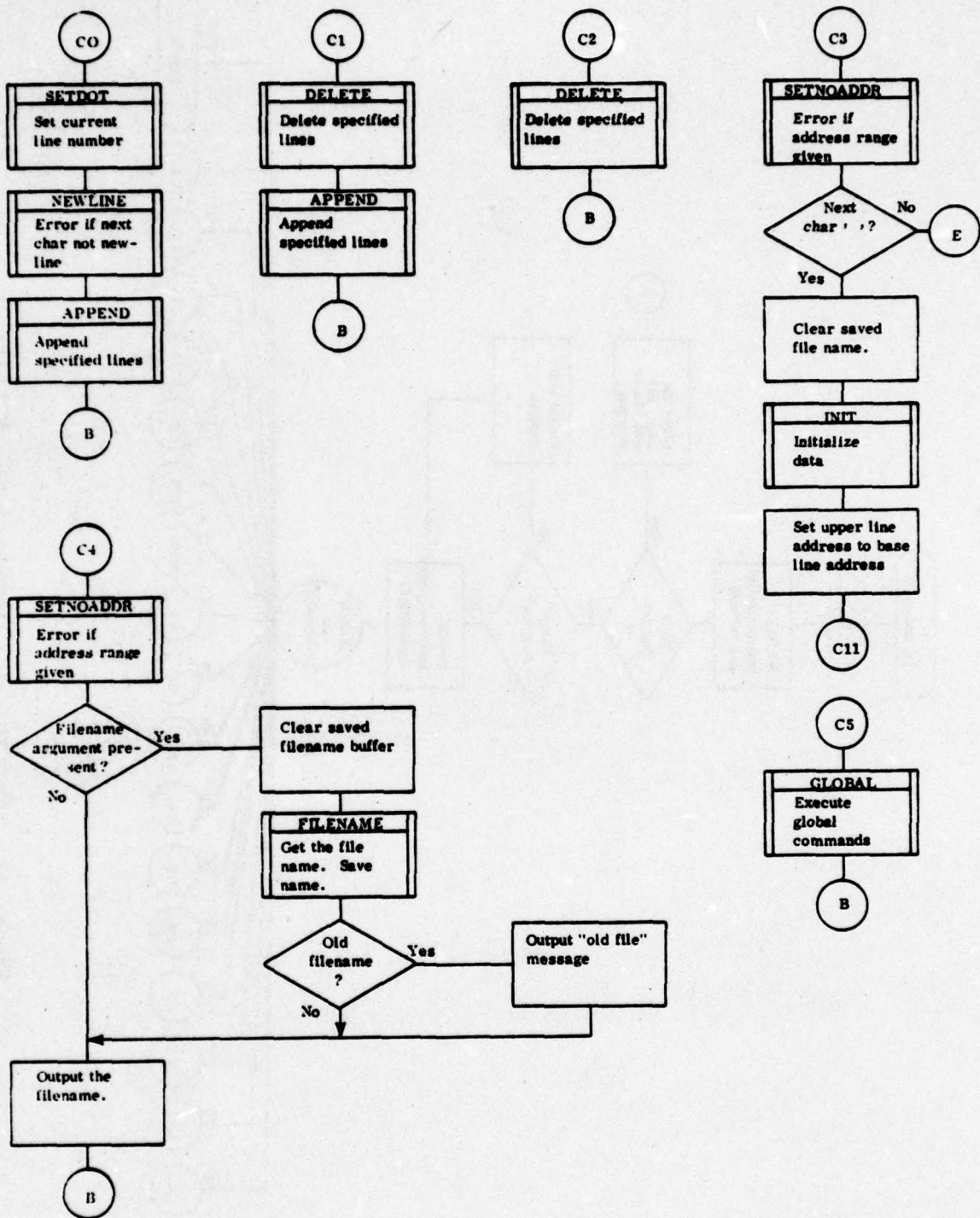


Figure 22. EDIT Data Flow (Sheet 3 of 7)

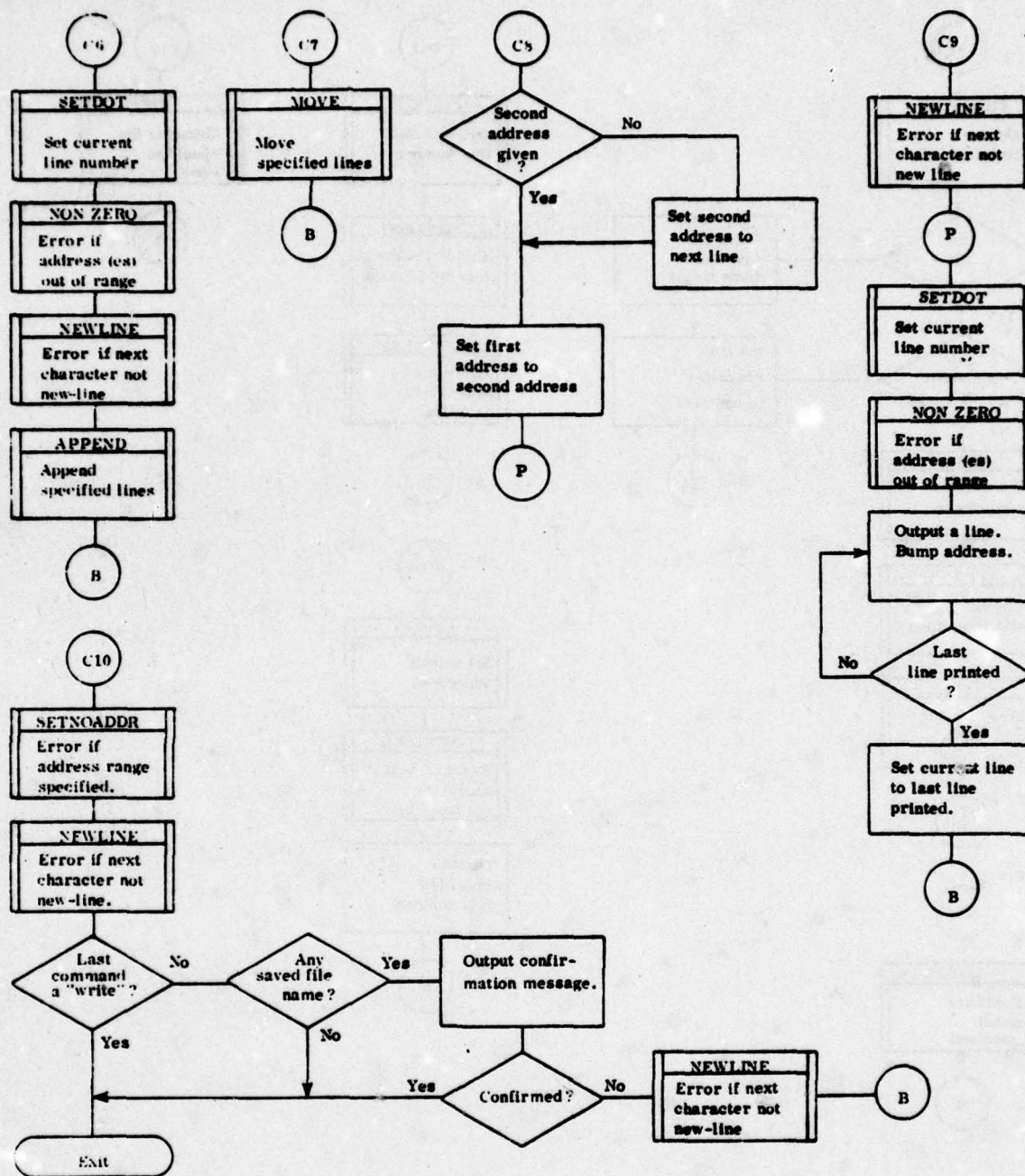


Figure 22. EDIT Process Data Flow (Sheet 4 of 7)

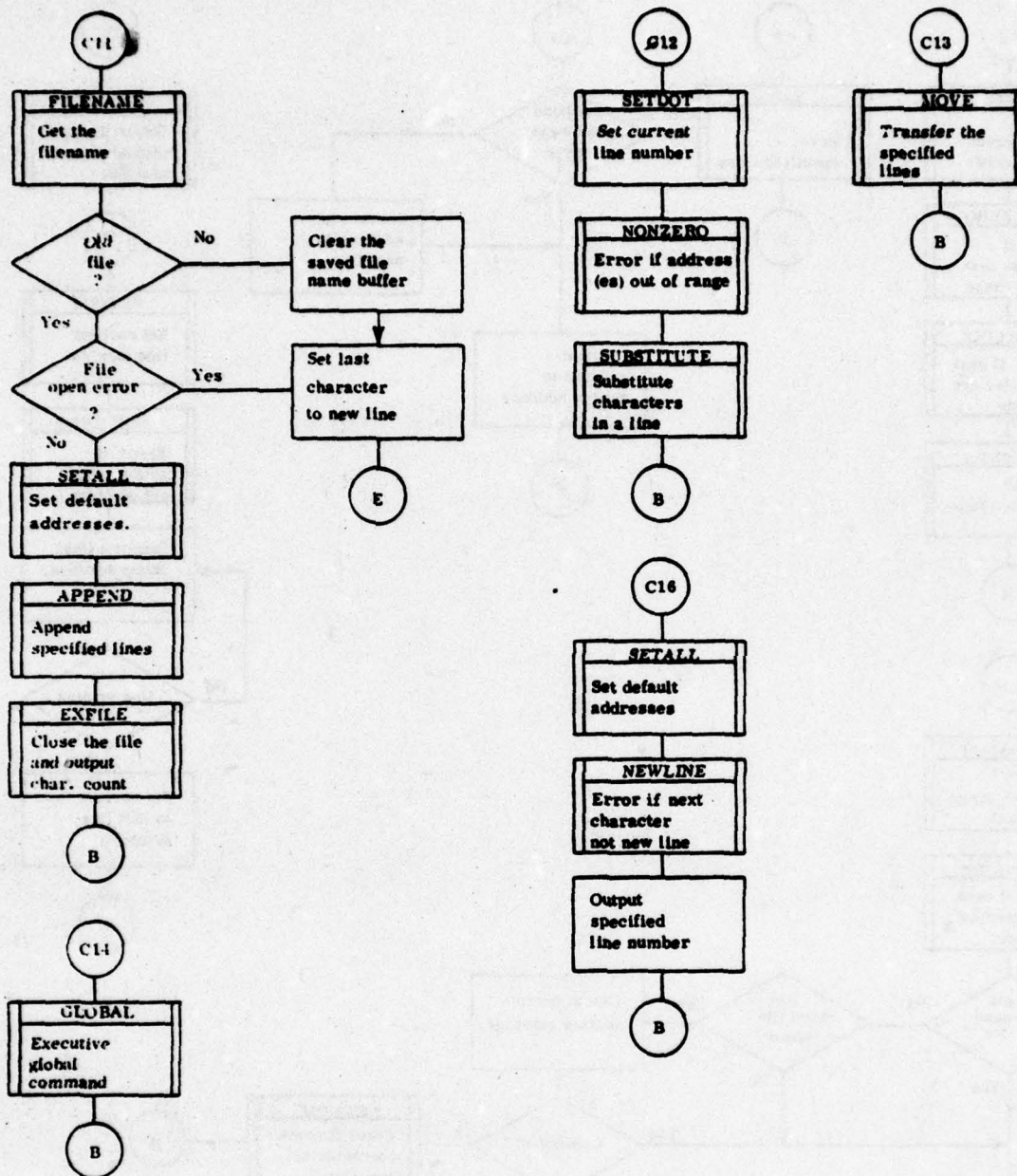


Figure 22. EDIT Process Data Flow (Sheet 5 of 7)

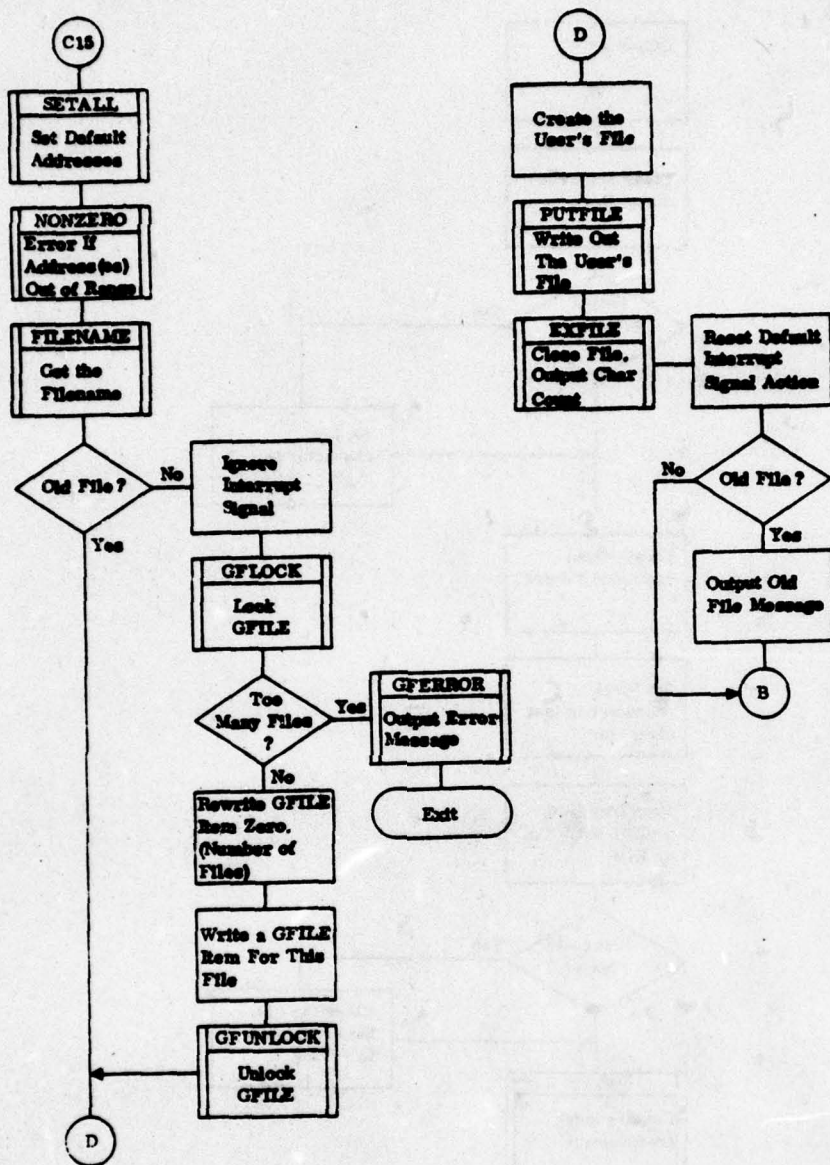


Figure 22. EDIT Process Data Flow (Sheet 6 of 7)

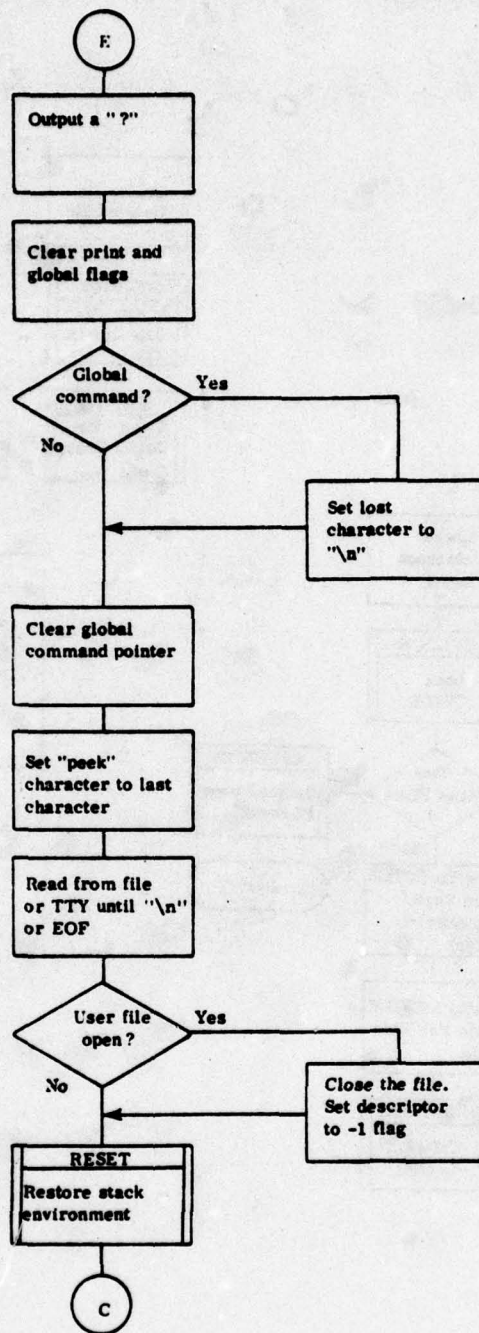


Figure 22. EDIT Process Data Flow (Sheet 7 of 7)

FILIST (List User's Files)

The FILIST process processes the FILIST command. It lists all users' text and query/response files for the TASMASTER. The TASMASTER has the option to specify a given TAS user whose files he wishes to peruse.

GLOBAL DATA USAGE - The FILIST process uses the following global data:

GBQST - user batch query statuses

GFILE - user text file descriptions

GUDES - user descriptions

LOCAL DATA USAGE - The FILIST process uses two buffers of significance.

FIIBUF - Buffer structure to read in the statistics for a given file.

FILBUF - Structure to store a line of information for a file. The file's name, block size, time of last access or time when the query was submitted/responded to, and query status are stored in the buffer.

GENERAL PROCESS FLOW - The FILIST process initially checks to ensure that the user is the TASMASTER. If not, an error message is output and the process exits. If a single user is specified, a flag is set accordingly. Next FILIST opens and reads in GUDES, one item at a time. For each non-empty item of GUDES, the following procedure is executed.

1. If a user is specified by the TASMASTER, a match is attempted between the GUDES SNI and the input SNI. If the match is unsuccessful, the next GUDES item is read.

2. If, however, the match is successful or a single user was not specified, the user sni information is output and the user's path-names are established.
3. The function FIFILEF is called to output the user's text file data.
4. The function FIBQST is next called to display batch query jobs' statistics.
5. If just the single user was specified, the process FILIST exits; otherwise, the next GUDS entry is read.

Having reached the end-of-file for GUDS, a last check is made. If the user was specified but not found, an error message to that effect is output. In either case, the process FILIST exits. (See figure 23 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

FIFILEF (List User's Text File Data) - FIFILEF initially reads GFILE and allocates the necessary storage space for the buffer FILBUF. For each non-empty item of GFILE, the function retrieves the file statistics and size, and stores the information in buffer FILBUF. Having collected this information, FIFILEF sorts the text files by access time. The text file statistics are then output sorted by last access time. The buffer storage is then freed.

If the GFILE file is empty, the TASMATER is informed that the user has no files defined.

FIBQST (List User's Batch Query Files) - FIBQST and FIFILEF perform logically the same function except for the files that are listed. FIBQST reads the first item of GBQST. If the file is empty, the TASMATER is so informed. Otherwise, FIBQST allocates the necessary storage for the buffer FILBUF and for each non-empty item in GBQST,

FIBQST retrieves the query and response files' statistics and size. The information is stored in FILBUF. Having collected the data, the query data are sorted by last submission/response time. The batch query/response files' data are then displayed for the TASMASTER. The function FIBQST frees the buffer storage and returns.

Report 76-C-0899-6

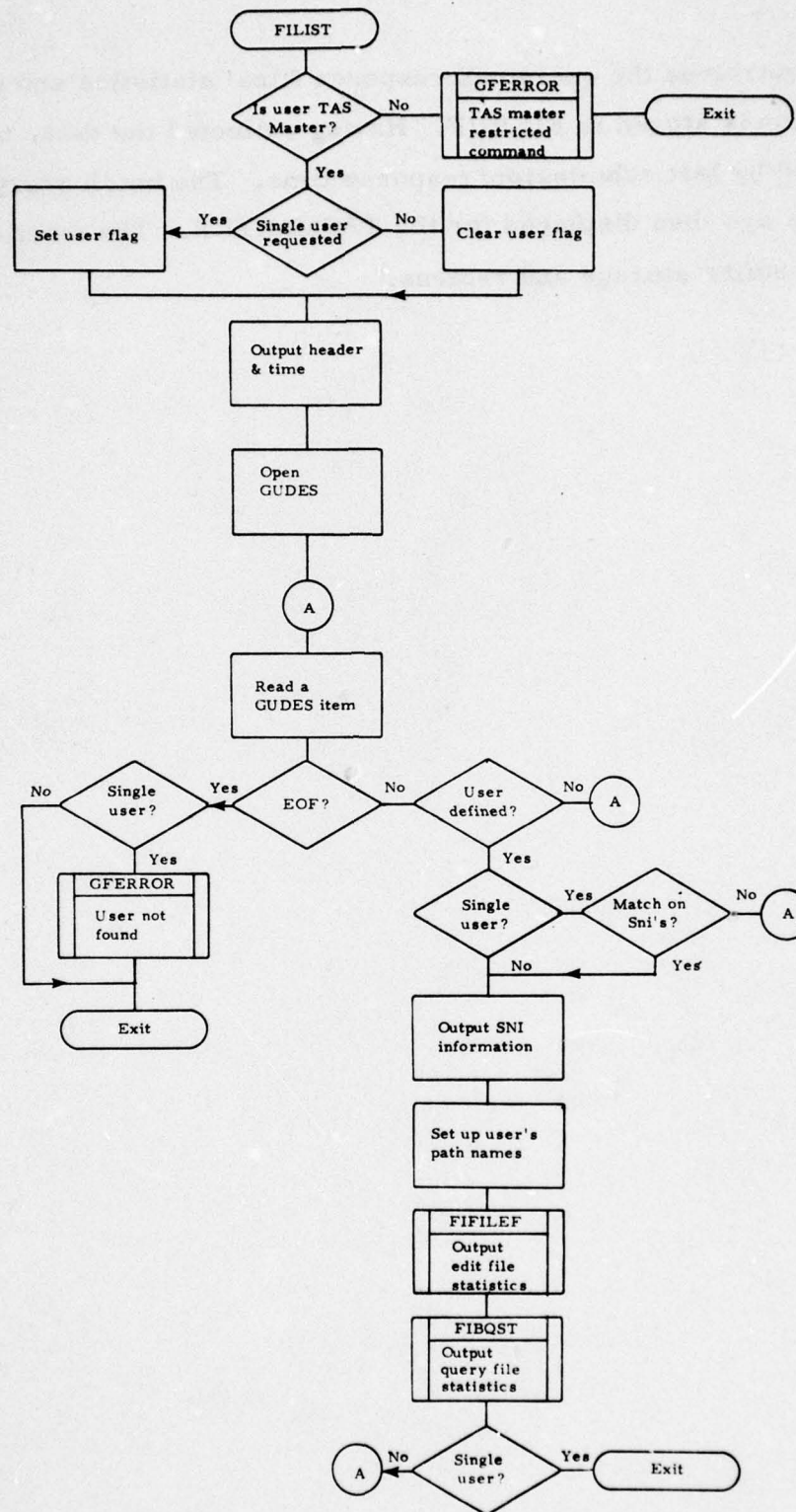


Figure 23. FILIST Process Data Flow (Sheet 1 of 3)

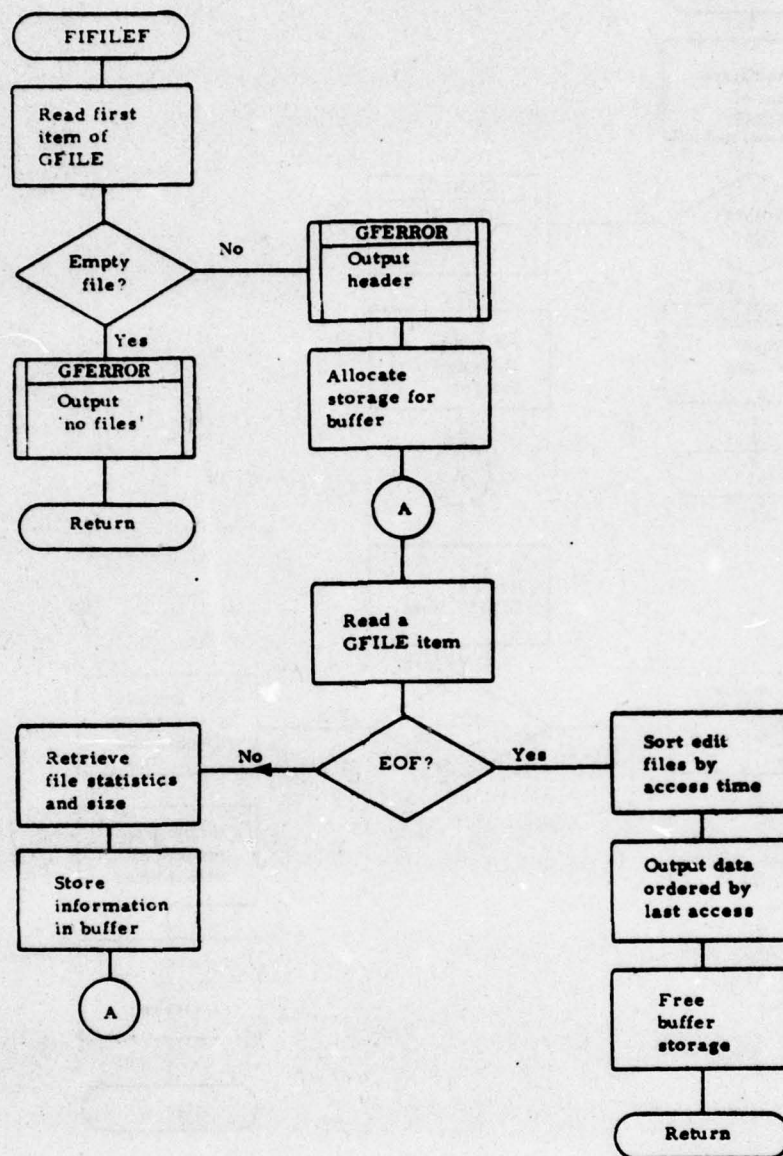


Figure 23. FILIST Process Data Flow (Sheet 2 of 3)

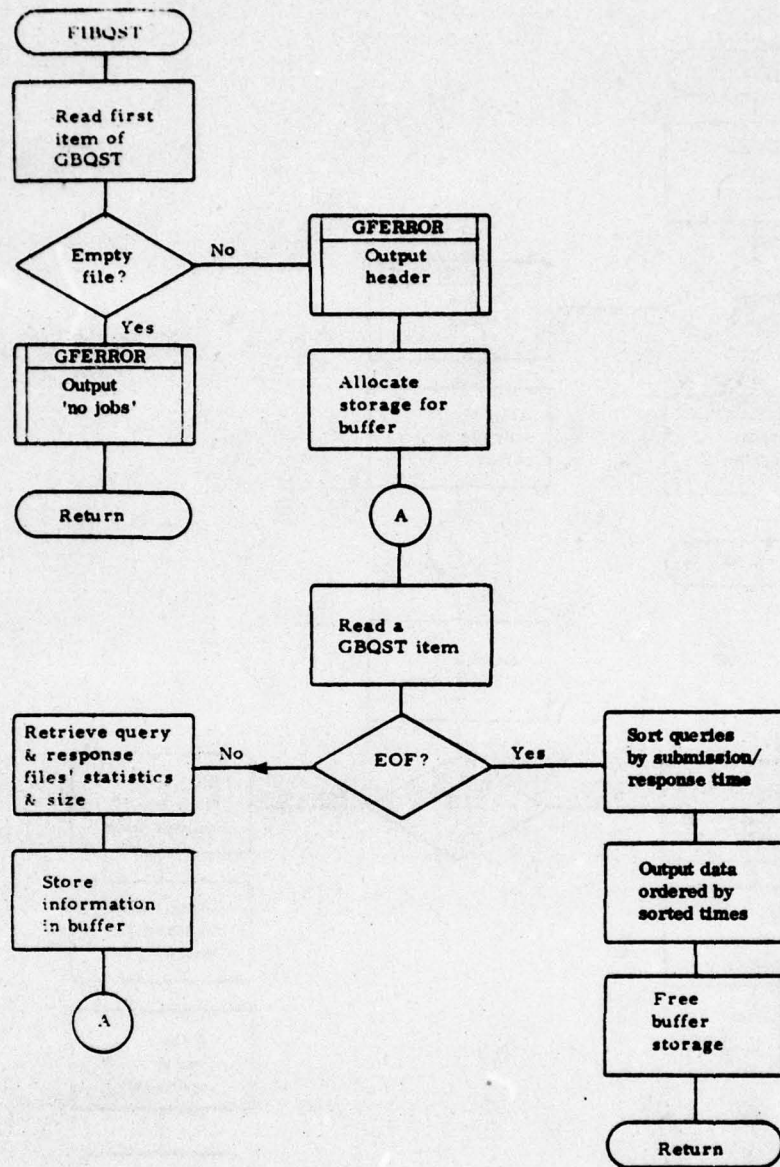


Figure 23. FILIST Process Data Flow (Sheet 3 of 3)

HELP (Output a Help File)

The HELP process processes the HELP command. It is responsible for displaying a "help" file which will assist the user in the execution of any of the TAS resources available to him. HELP may be called directly by the user, or indirectly by LOGON to assist the user in logging onto TAS.

GLOBAL DATA USAGE - HELP uses no pertinent global data.

LOCAL DATA USAGE - HELP uses no pertinent local data.

GENERAL PROCESS FLOW - HELP first checks whether the first argument begins with '-'. If so, the call was an internal call from LOGON and the command name is set to "logon".

If the first argument is not '-', the command name is set to the argument given or to "all" if no argument was specified.

The help file pathname is built and the file is output to the user's terminal with appropriate paging.

Finally, if HELP was called from LOGON, it sleeps for 20 seconds before terminating. (See figure 24 for data flow.)

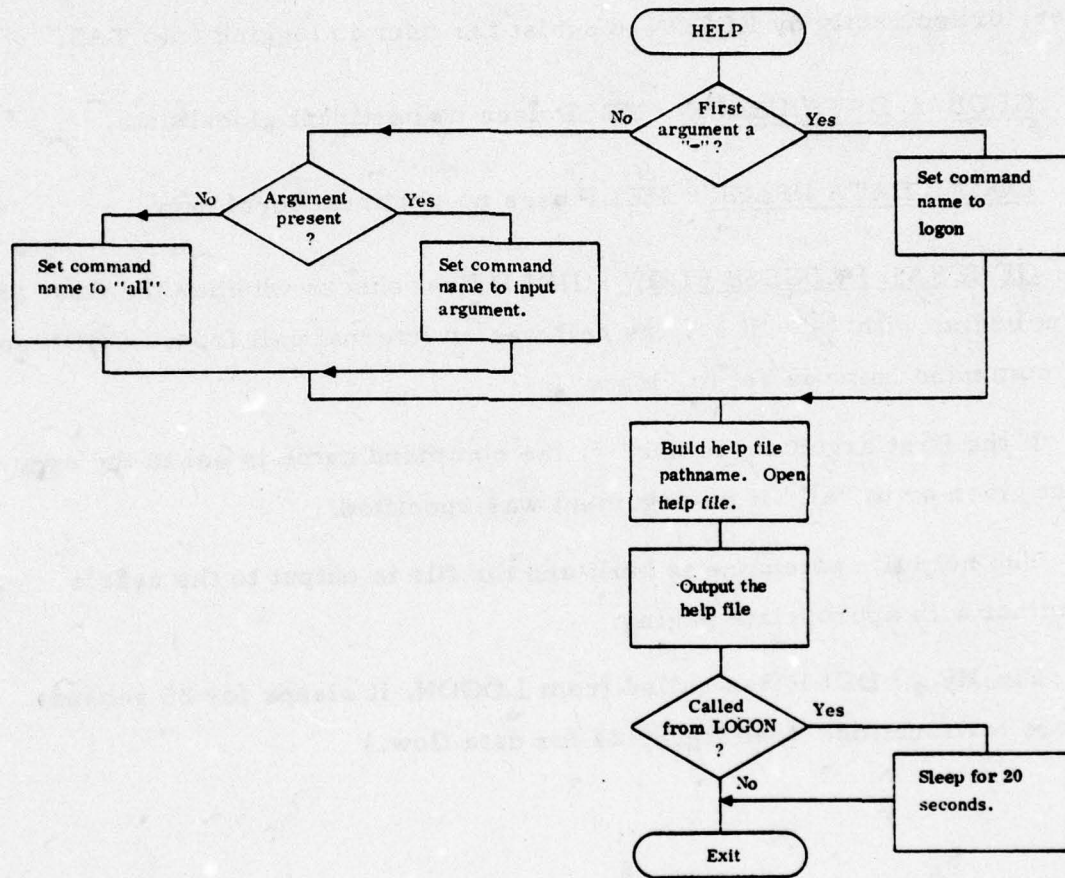


Figure 24. HELP Process Data Flow

HOLD (Hold One or All Batch Query Jobs)

The HOLD process processes the HOLD command. It allows a user to cause one or all of his batch query jobs to be "held", protecting them from automatic deletion (if they have been delivered) when he logs off of TAS.

GLOBAL DATA USAGE - HOLD uses the following global data:

GBQST - user batch query statuses

LOCAL DATA USAGE - HOLD uses no pertinent local data.

GENERAL PROCESS FLOW - HOLD first checks whether any arguments have been input. If there are no input arguments, an error message is output and HOLD terminates. Otherwise, if the input argument is "ALL", an all jobs flag is set.

HOLD locks GBQST, opens it and reads the first item. If the user has no jobs, a message is output and HOLD terminates.

If the user has one or more jobs, the job count is saved and GBQST items are read and processed until end-of-file. For each occupied GBQST item, if the "all" option was specified, HOLD sets the job status to hold and rewrites the GBQST item; otherwise, if the JOBID matches the input JOBID, HOLD sets the status of the specified job to hold and terminates.

If processing continues until end-of-file, HOLD checks to see if a JOBID was specified. If so, an error message is output and HOLD terminates. Otherwise, the "all" option was specified and if no jobs were held, a message is output and HOLD terminates. (See figure 25 for data flow.)

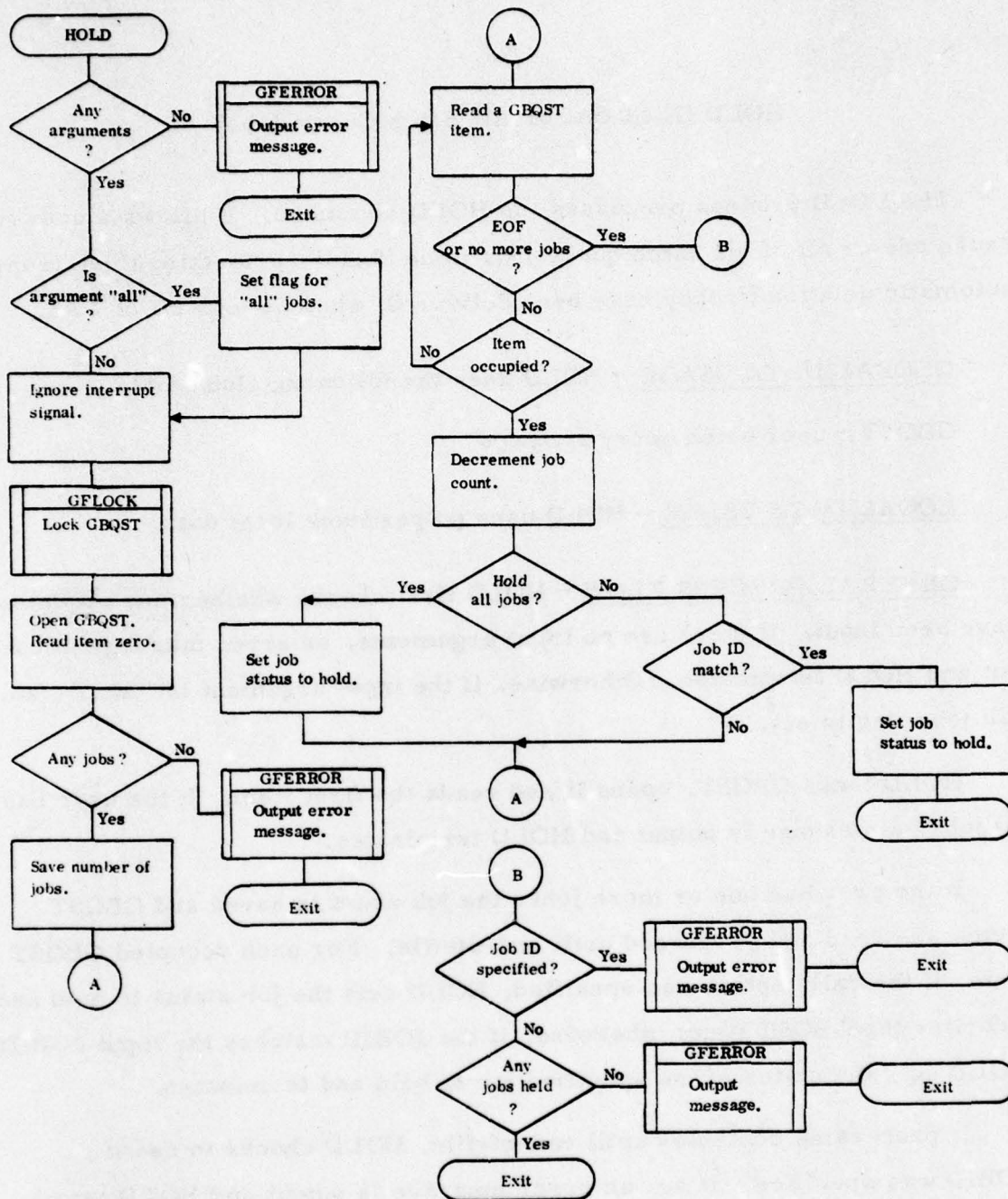


Figure 25. HOLD Process Data Flow

INI (Intelligent Network Interface)

The INI process is responsible for emulating the COINS I protocol on the COINS II network. INI builds COINS I messages to be sent across the network, processes RCPTs for those messages, and retransmits messages when required. INI also strips received COINS I messages and passes the resulting messages to BQRD for system processing.

GLOBAL DATA USAGE - INI uses the following global data:

- GAPNM - application names
- GHOST - host descriptions
- GIMSG - INI batch message descriptions
- GTCTS - network message traffic counts

LOCAL DATA USAGE - INI uses the following local data:

- INADDR - addressee application names
- INAST - ABRT sub-types
- INMTYP - COINS I message types
- INSTYP - SRVC sub-types

GENERAL PROCESS FLOW - INI is called with a varying number of input parameters. The first parameter is always a function code. The second parameter is an application (host) ID except for the message timed-out function, where the second parameter is the JOBID of the timed-out message. The third parameter for those functions specifying output to the network is a JOBID.

INI first picks up the function code and host ID, if present. If the function specifies output to the network, then the following processing is performed. INI locks its batch message descriptions (GIMSG) file via GFLOCK and then opens GIMSG. INI reads and searches GIMSG to determine whether the JOBID is already present. As this is highly unlikely, if not impossible, if INI finds the JOBID, INERROR is called to exit. INI reads and searches GIMSG again for the next empty item. The job's function code, JOBID, transmission time, host ID, and transmission count of one are stored in GIMSG. If the job has a data file, the pathname of the data file is saved. INI then calls INFMTCONS to build the COINS I message and send it through the network. INERROR is then called to exit.

If the function code does not specify output to the network, then it specifies either a message received, a message timed out, or a network read error. INI switches on the function code. If the function code specifies a network read error, then INI sets a BQRD function code to network read error and sets a BQRD JOBID parameter to zero. INHSTDN is then called to delete the host's INI jobs. INI then performs an execute sequence to call BQRD with the network read error.

If the function code is a message timed out, then INI locks GIMSG via GFLOCK and opens GIMSG. INI reads and searches GIMSG for the JOBID of the timed-out message. If the JOBID is not present (i.e. a RCPT was received for it), then INERROR is called to exit. Otherwise, INI picks up the message's host ID from GIMSG in preparation for retransmission of the message.

The INI retransmission processing is as follows. INI sets up the maximum transmission count for the message (two times for SRVC (CHECK),

nine times for others). The message's transmission count in GIMSG is incremented and then tested against the maximum count to see if the message can be sent again. If the message has already been sent the maximum number of times, then the message must be logged out as follows. First INI calls INCTS to update the network message traffic counts (GTCTS) with a NO REPLY. The segment count of the message and its JOBID are saved as BQRD parameters. The BQRD function code is set to message logged out. The GIMSG entry for the message is deleted by being set to zeroes. The INI data file containing the formatted COINS I packets for the message is unlinked. GIMSG is written and then unlocked via GFUNLOCK. INI then performs an execute sequence to call BQRD with a message logged out. If the message is not to be logged out, but can be retransmitted, then INI stores the new transmission time in GIMSG. If this is the first retransmission of the message, then INI opens the data file containing the message's COINS I packets, positions the file to the retransmit flag, writes a retransmit flag (R), and then closes the data file. INI writes GIMSG and calls GFUNLOCK to unlock GIMSG. INSEND is called to send the message through the network and INI exits.

If the function code specifies a message received, then INI links the Read Daemon's (RD) temporary file to an INI temporary file (created with a unique name containing the INI process ID). The Read Daemon's temporary file is then unlinked and also unlocked by GFUNLOCK. INEXTCNS is called to extract a message from the received COINS I packets and returns either a message type or an error value. If an error value is returned, then INERROR is called to exit. If a message type is returned, then INCTS is called to update GTCTS with the appropriate message type received. INI then switches on the message type received. All possible COINS I messages

that are utilized on the COINS II network are processed by INI including SRVC (ALERT)s which are no longer used. The processing performed for each message type is described below.

If the message type is either an INTG or a SRVC(TRACE), neither of which is processed by TAS, then INI unlinks the network response file (if any), performs a fork/execute sequence to call the TAS Logging Process (TLP) with a bad JOBID, and calls INERROR to exit.

If the message type is RLSE, then INI sets a BQRD function code to RLSE and saves the message's network ID for BQRD.

If the message type is ANSR, ABRT (FAULT or CAUSE), then INI sets a BQRD function code to a corresponding value and saves the classification (only meaningful for ANSR) and segment count of this response for BQRD.

If the message type is SRVC(TRACK), then INI sets a BQRD function code to SRVC(TRACK) and saves the trace results for BQRD.

If the message type is SRVC(PRINT), then INI sets a BQRD function code to SRVC(PRINT) received and sets a BQRD JOBID parameter to zero.

If the function code is SRVC(BREAK), SRVC(ALARM), SRVC(READY) original or response, then INI sets the BQRD function code to a corresponding value and sets a BQRD JOBID parameter to zero. If the message is a SRVC(BREAK), then INHSTDN is called to delete the INI jobs for the specified host.

The remaining message type is RCPT, and since this is processed differently from the other messages, it is discussed at the end of this section.

For the above message types, with the exceptions, as noted, of INTG, SRVC(TRACE), and RCPT, INI finishes building the prestored RCPT as

extracted by INEXTCNS. The end of message data and a segment count of one are stored in the RCPT. INI then creates a data file, writes out the RCPT, and closes the data file. INSEND is called to send the RCPT through the network. After RCPTing the message, INI again switches on the message type for internal processing. If the message is an ABRT(ERROR) or a SRVC(ALERT), both of which reference a message which was invalidated in some way at the host, then INI locks GIMSG via GFLOCK and opens GIMSG. INI reads and searches GIMSG for the referenced JOBID. If the JOBID is not found, then INERROR is called to exit. Otherwise, INI calls GFUNLOCK to unlock GIMSG and INI goes to the retransmission processing described earlier with the timed out message logic.

If the message is a SRVC(READY) original, then INI opens, reads and closes the Host Descriptions (GHOST) file. INEXTJB is called to get the next JOBID and to set up an item for a job in GIMSG. If the TAS status is down in GHOST, then a function code is set to SRVC(BREAK), otherwise a function code is set to SRVC(READY) response. INFMT CNS is called to format and send a COINS I SRVC(BREAK) or SRVC(READY) response message through the network.

After the internal processing described above, INI determines whether a BQRD function code has been established for any of the received message types. If there is a BQRD function code, then INI performs an execute sequence to call BQRD. Otherwise INI exits.

RCPT processing, as noted previously, is somewhat different than for other received message types. INI locks GIMSG via GFUNLOCK and opens GIMSG. INI then reads and searches GIMSG for the JOBID being RCPTed. If the JOBID is not present, then INERROR is called to exit. If the JOBID is present, then the message type of the JOBID is picked up from GIMSG and

the message's segment count is saved for TLP. The message's GIMSG entry is deleted by setting it to zeroes and the message's INI data file is unlinked. INI writes GIMSG and calls GFUNLOCK to unlock GIMSG. INI then switches on the message type of the RCPTed job. If the message type is INTG, INI performs a fork/execute sequence to call TLP with an INTG RCPT received and INERROR is then called to exit. If the message type is SRVC(PRINT), SRVC(READY), or SRVC(BREAK), then a BQRD function code is set to the corresponding RCPT received value. If the message being RCPTed is a SRVC(BREAK), then INHSTDN is called to delete this host's jobs. INI then performs a fork/execute sequence to call BQRD with the appropriate RCPT received function code. (See figure 26 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

INFMTCONS (Format and Send a COINS I Message) - INFMTCONS is called with a message type, a JOBID and a saved pathname of a file of text to be stored in the message (if applicable). INFMTCONS creates the INI data file for this message and then stores the necessary fixed header data in the first segment of the message. The appropriate message type from INMTYP and the recipient host's trigraph from INADDR are stored in the header as well as the TAS as originator trigraph, 'TAS'. If the message type is not an INTG, then the appropriate SRVC subtype from INSTYP is also stored in the header. If there is a data file specified, then INFMTCONS reads blocks of data and builds message segments, writing out each message segment after it is built. After the data has been stored, or if there is no data, then INFMTCONS stores the end of message data and a segment count in the final segment (may be the first segment as well). The segment count is stored in GIMSG and GIMSG is written out to disk. GFUNLOCK is then called to

unlock GIMSG. The message's final segment is written to the INI data file and INSEND is called to send the job through the network. INFMTCONS finally calls INCTS to update GTCTS and returns.

INEXTJB (Set up an INI Batch Job) - INEXTJB first calls GFJOBID to get the next sequential JOBID. GFLOCK is then called to lock GIMSG and GIMSG is opened. INEXTJB reads and searches GIMSG for the next empty item. The job's transmission time, JOBID, host ID, and transmission count of one are stored in GIMSG, and INEXTJB returns.

INEXTCNS (Extract Message from COINS I Packets) - INEXTCNS opens INI's uniquely named temporary file containing the network response data and then unlinks the temporary file so that it will be removed from the system after it is closed. The first segment of the message is then read. INEXTCNS masks out the odd parity bits for each character in the segment and keeps a count of parity errors encountered. INEXTCNS then tries to make a match against a message type of 'RCPT'. If the message type is RCPT, then INEXTCNS picks up the JOBID being receipted and returns with a message type of RCPT. INEXTCNS picks up and saves the message's pre-stored RCPT in another buffer. The network ID and referenced JOBID, if present, are picked up and saved. INEXTCNS then tries to make a match against the other message types in INMTYP. If the message type is invalid, then INEXTCNS returns with an error value. Otherwise the message type is saved. The addressee trigraph is validated to be 'TAS' and the originating host's trigraph is validated against the corresponding value in INADDR. If either value is invalid, INEXTCNS returns with an error value. If the message type is SRVC, then INEXTCNS tries to make a match of the subtype against the values in INSTYP. If the subtype is invalid, INEXTCNS returns

with an error value. If the subtype is valid, then the message type is adjusted to reflect the subtype. The single character receipted message type in the prestored RCPT is checked to see if the SRVC message is a SRVC(READY) response. If it is, then the message type is set to SRVC(READY) response. If, however, the message type is not SRVC, but is ABRT, then INEXTCNS tries to make a match of the subtype against the values in INAST. If the subtype is invalid, INEXTCNS returns with an error value. Otherwise, the message type is adjusted to reflect the subtype. If the message type is ANSR, ABRT (FAULT or CAUSE), then INEXTCNS picks up the classification of the response. If the message type is SRVC(TRACK), then INEXTCNS picks up the trace results. If the message has additional data, i.e. is an ANSR, ABRT (FAULT or CAUSE), or SRVC(PRINT), then the following processing is performed. A network response file is created and the additional message data is extracted from the packets and written to the response file. If the data spans more than one segment, then each segment is read, the odd parity bits are masked out and a count of total parity errors is maintained. After the final segment is processed, INEXTCNS validates the segment count in the final segment. If the segment count is incorrect, or if there were more than ten parity errors per segment, then INEXTCNS unlinks the network response file and returns with an error value. INEXTCNS saves the pathname of the network response file. For all valid messages, INEXTCNS returns with the message type.

INSEND (Send a Batch Job) - INSEND first determines whether there is an open connection to the specified host by testing for the existence of the Write Daemon's (WD) process ID (PID) file for that host. If there is no open connection, then INSEND calls INHSTDN to delete this host's INI jobs, unlinks the INI data file for the job to be sent, and calls INERROR to exit.

If there is an open connection, then GFLOCK is called to lock the write socket. After the lock is performed, INSEND checks to see if the connection is still open. If not, then INSEND calls GFUNLOCK to unlock the write socket and then cleans up data and exits as described above. Otherwise, INEXTCNS opens and reads the write daemon's PID. A job pathname file is created and the pathname of the job's INI data file is written. A 'write' signal is sent to the write daemon and INEXTCNS returns.

INHSTDN (Delete a Host's Jobs) - INHSTDN calls GFLOCK to lock GIMSG and then opens GIMSG. INHSTDN reads and searches GIMSG for jobs which are for the specified host. For each such job, INHSTDN deletes the GIMSG entry by setting it to zeroes, writes the GIMSG item, and unlinks the job's INI data file. After all jobs for the host are deleted, GFUNLOCK is called to unlock GIMSG and INHSTDN returns.

INCTS (Update GTCTS) - INCTS is called with a message type either sent or received or NO REPLY. GFLOCK is called to lock GTCTS and INCTS then opens and reads GTCTS. For the message type sent or received or NO REPLY, the appropriate counter is incremented. Where applicable, a segment counts counter is also updated. INCTS writes and closes GTCTS, and GFUNLOCK is called to unlock GTCTS. INCTS then returns.

INERROR (Output Error and/or Exit) - INERROR is called with an error number or zero and an exit flag set to either zero or non-zero. If an error number is specified, then the specified error message is output via GFERROR. If the exit flag is not set, then INERROR returns. Otherwise, any locked files are unlocked via GFUNLOCK and INERROR exits with the specified exit value.

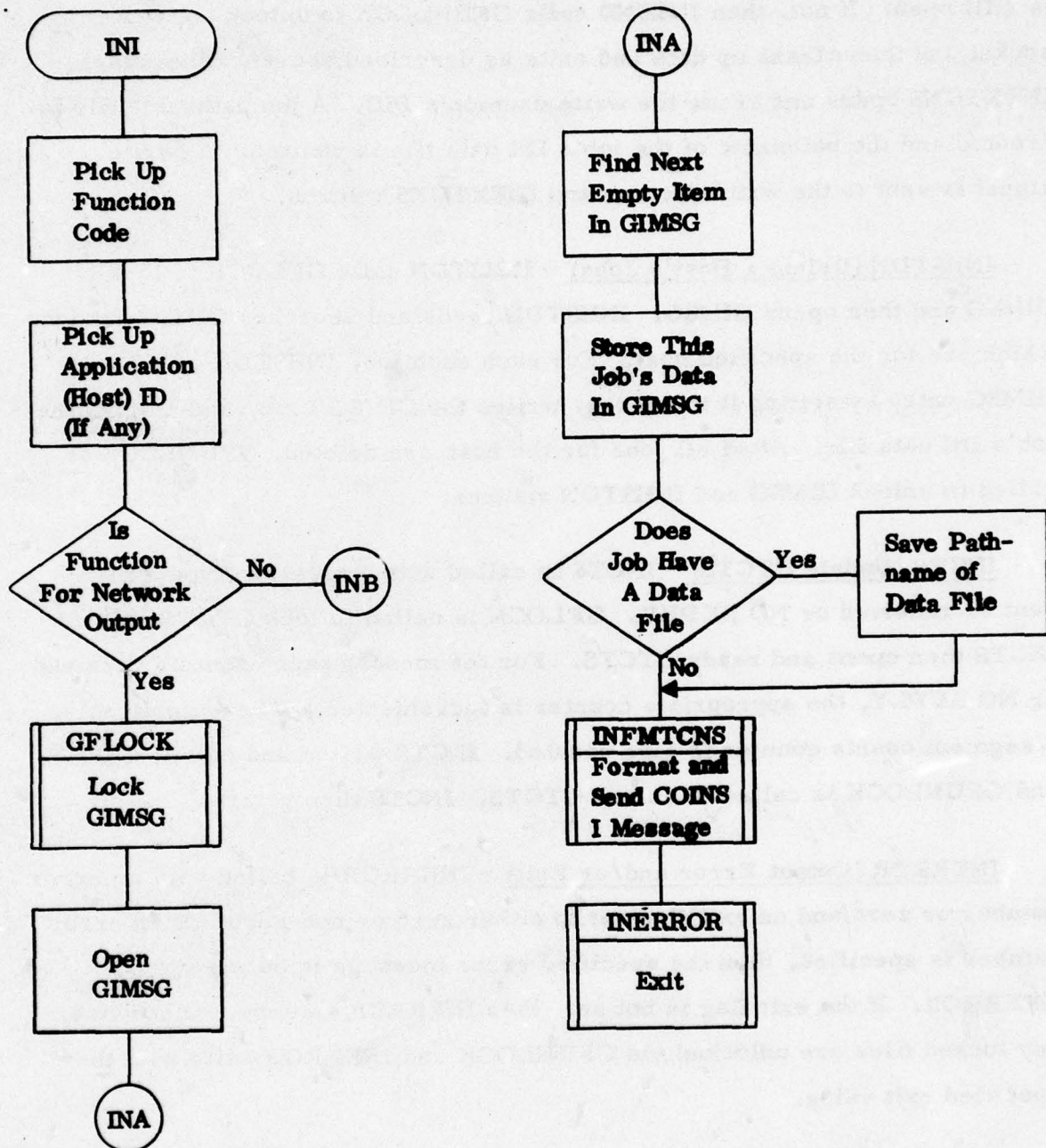


Figure 26. INI Process Data Flow (Sheet 1 of 18)

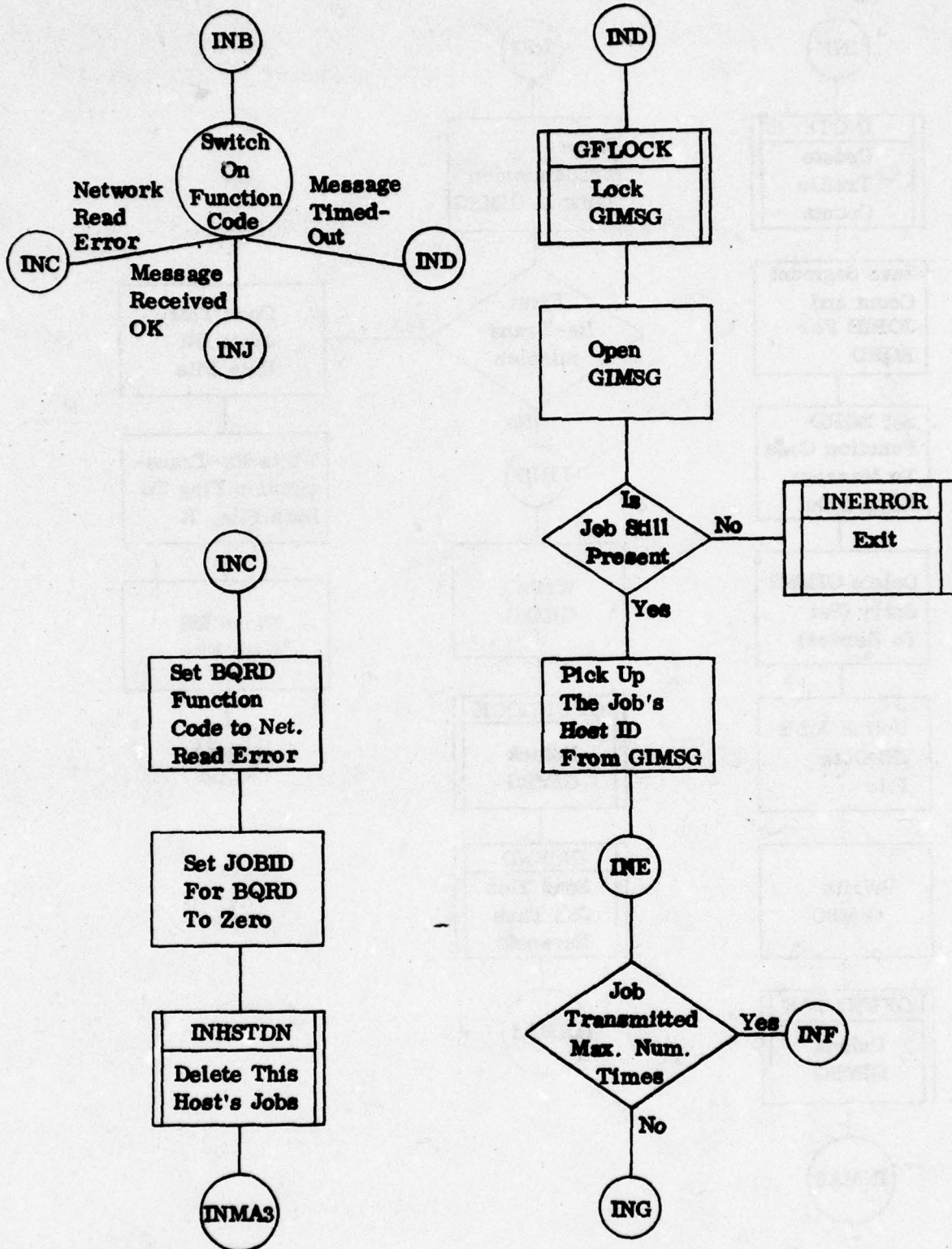


Figure 26. INI Process Data Flow (Sheet 2 of 18)

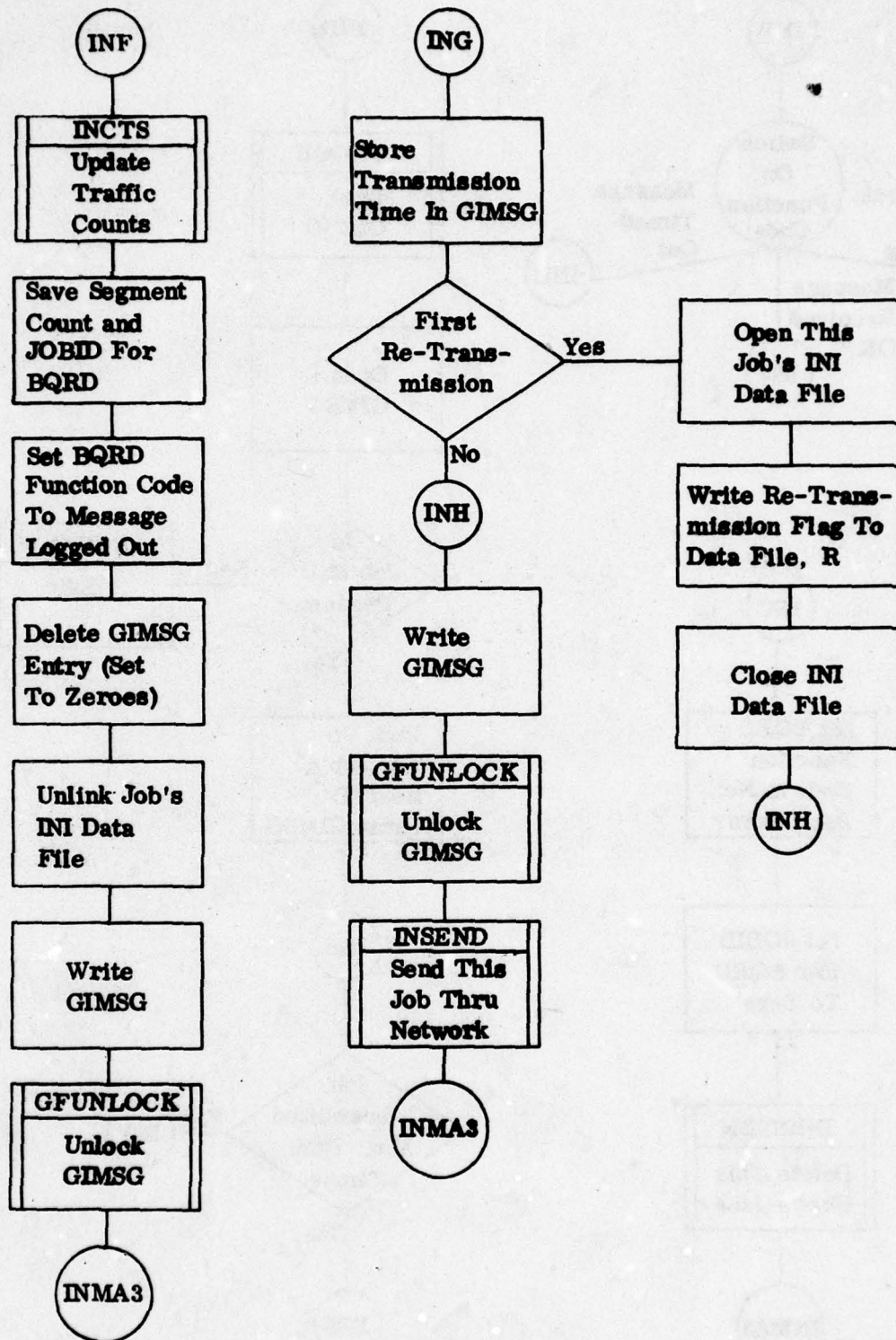


Figure 26. INI Process Data Flow (Sheet 3 of 18)

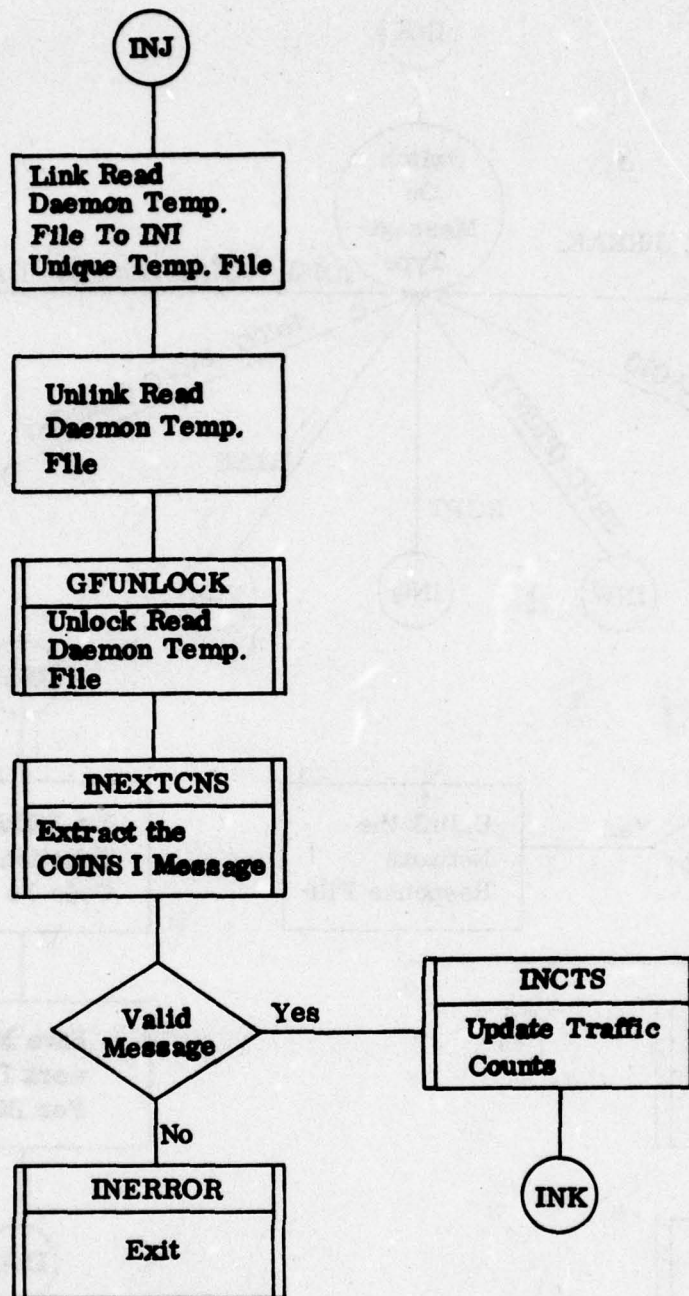


Figure 26. INI Process Data Flow (Sheet 4 of 18)

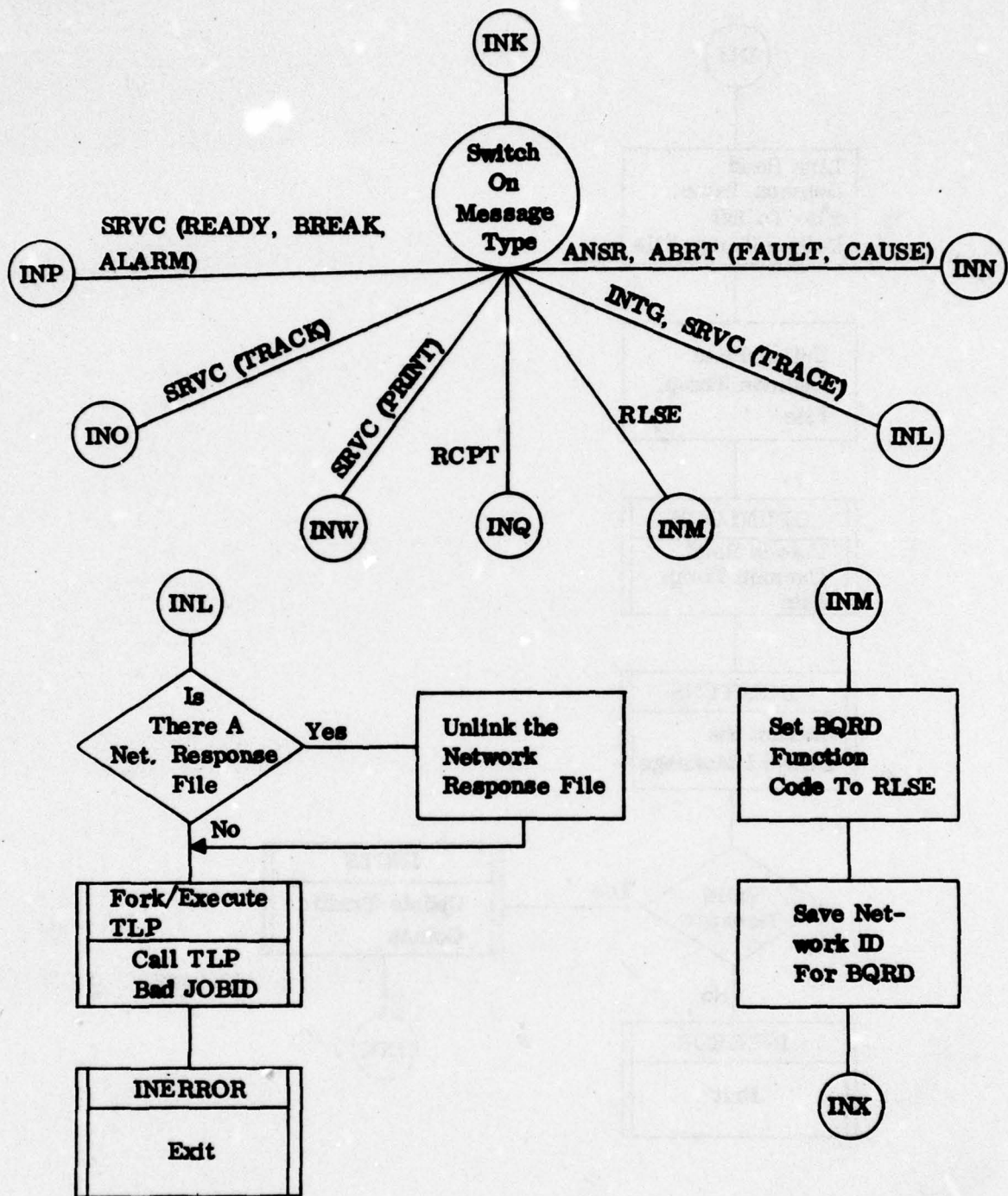


Figure 26. INI Process Data Flow (Sheet 5 of 18)

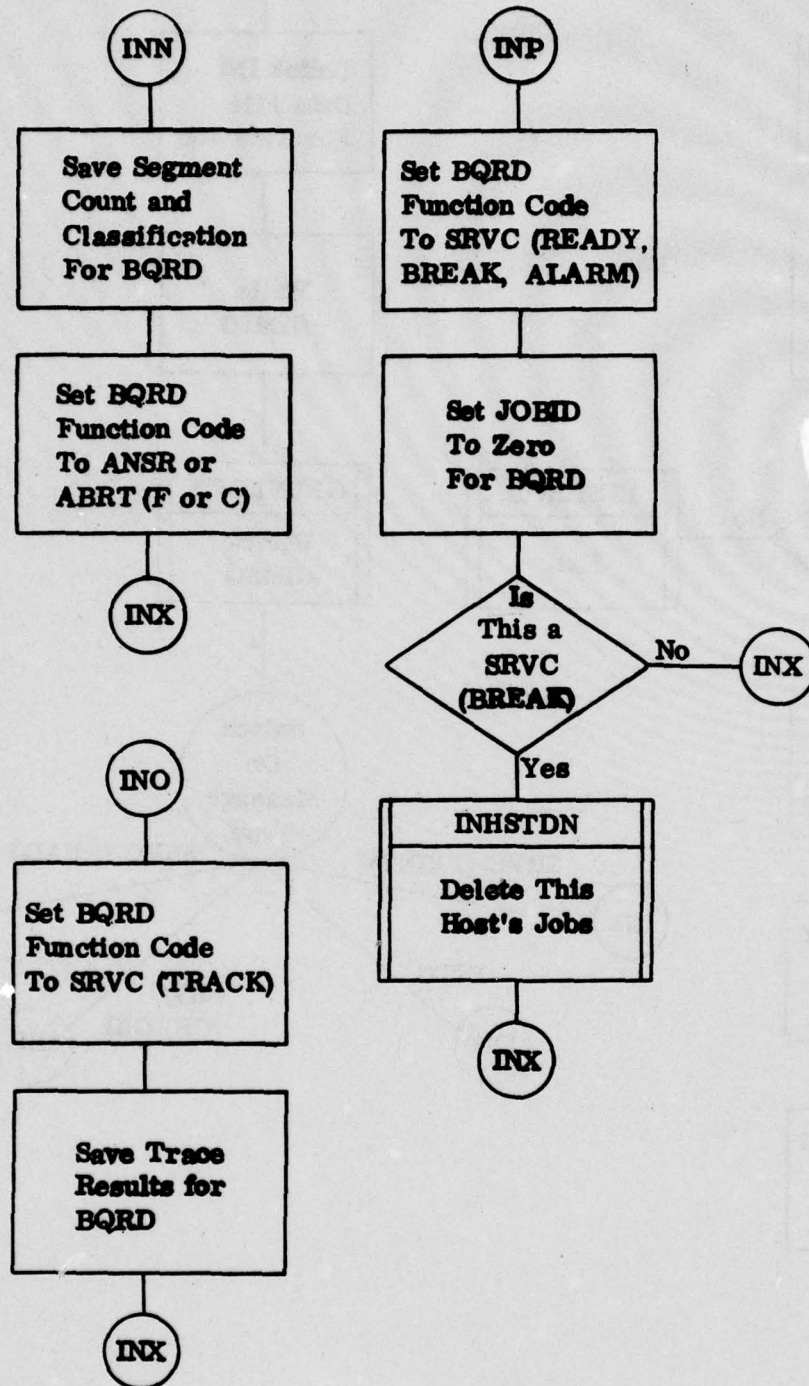


Figure 26. INI Process Data Flow (Sheet 6 of 18)

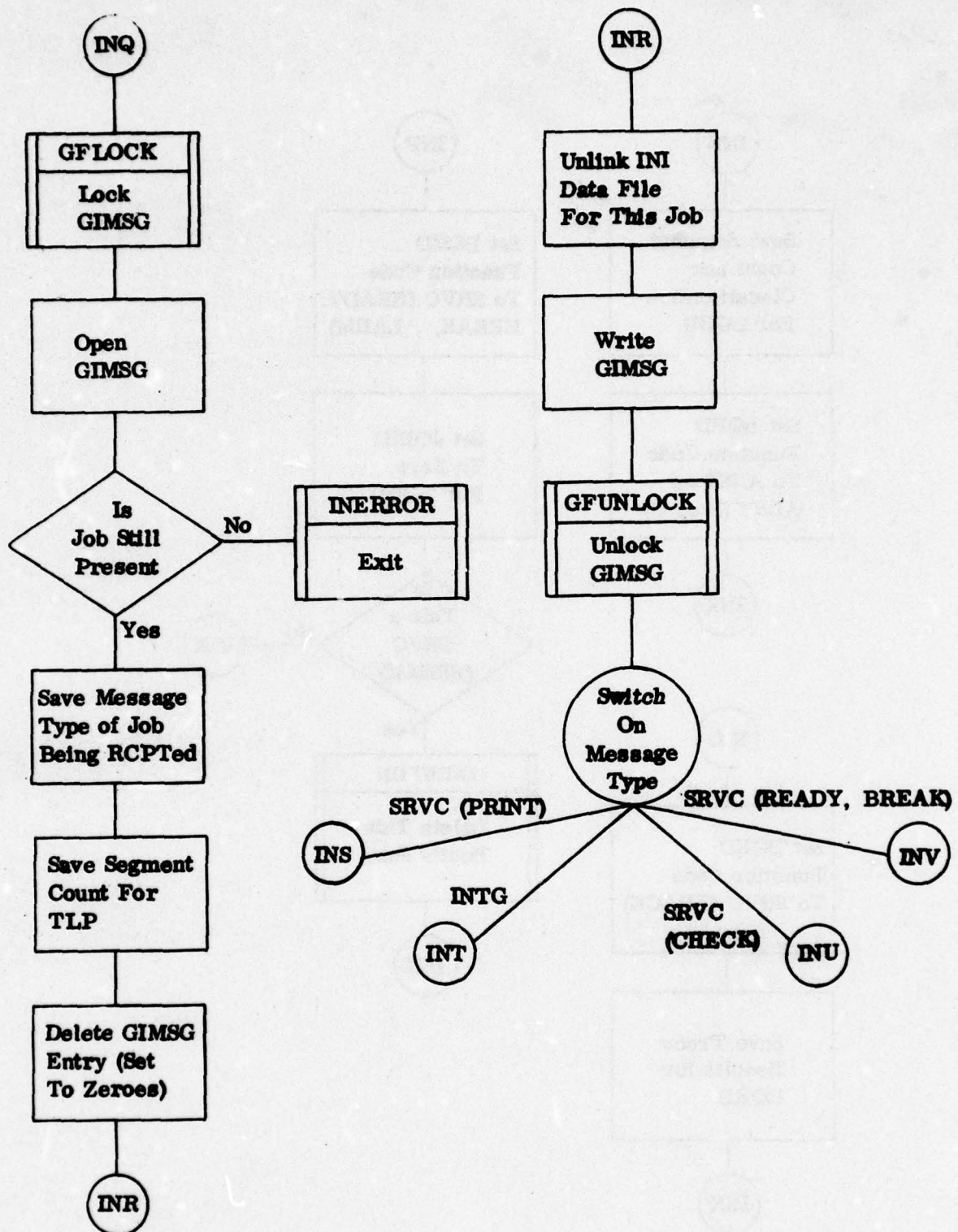


Figure 26. INI Process Data Flow (Sheet 7 of 18)

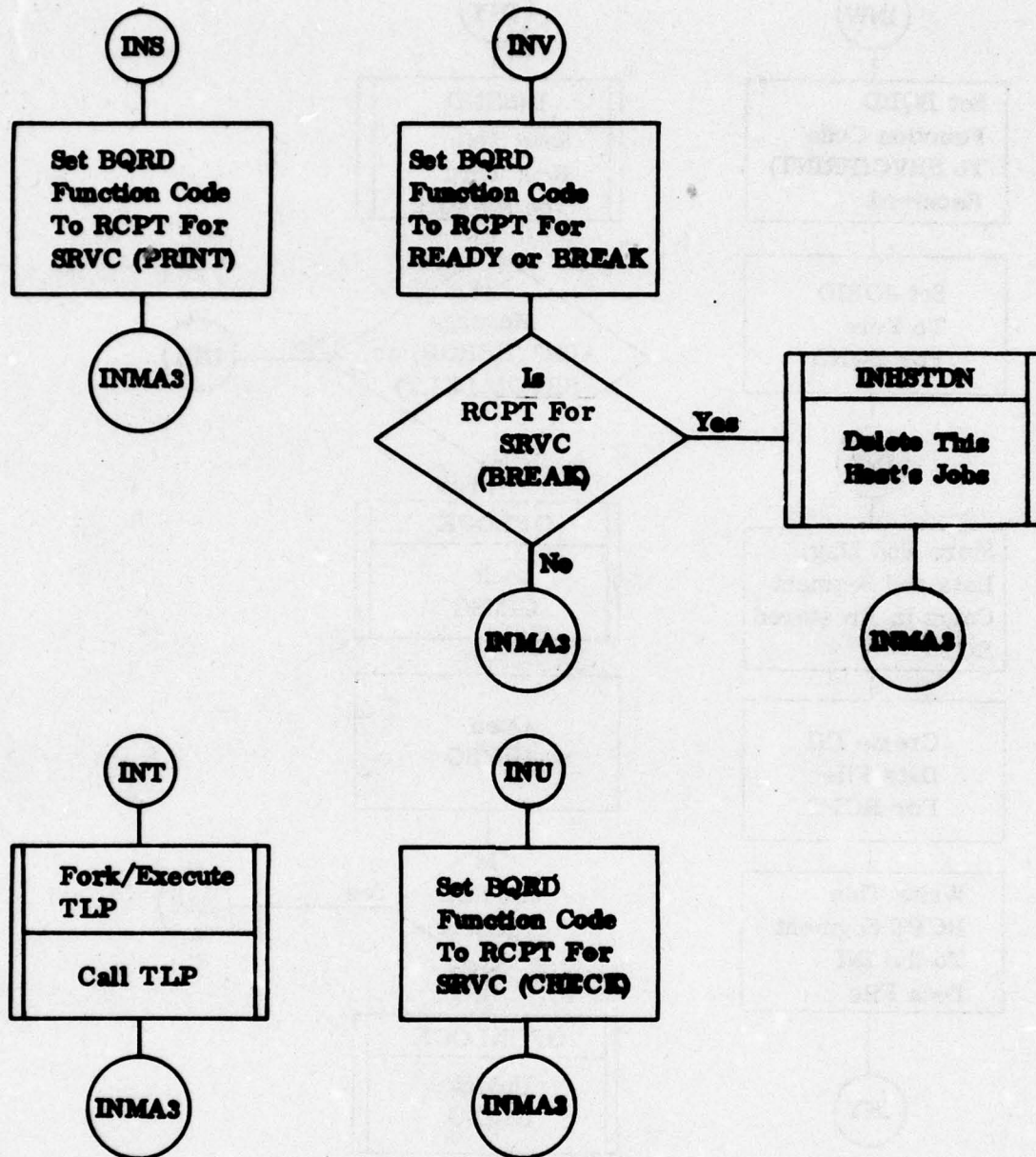


Figure 26. INI Process Data Flow (Sheet 8 of 18)

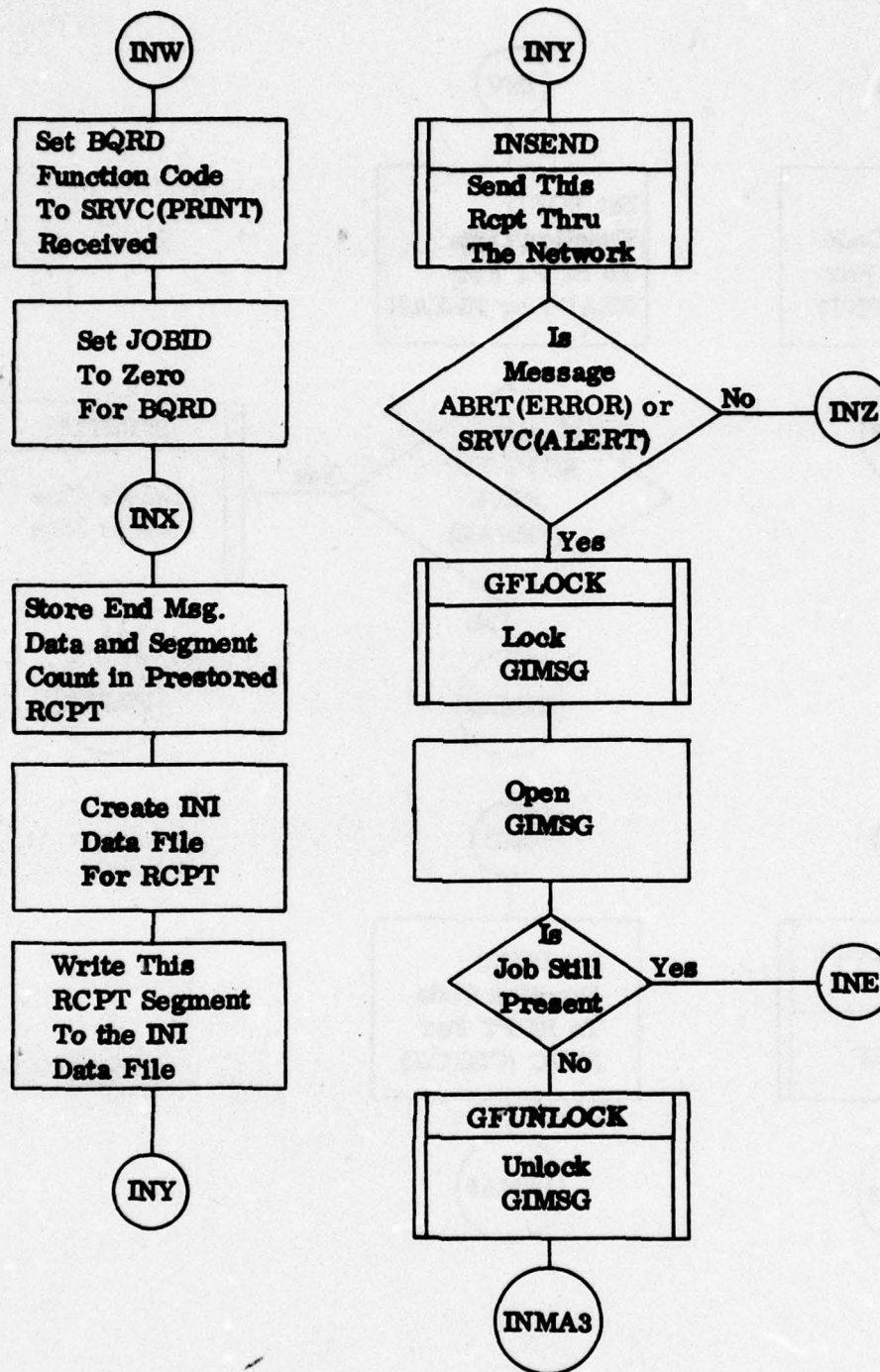


Figure 26. INI Process Data Flow (Sheet 9 of 18)

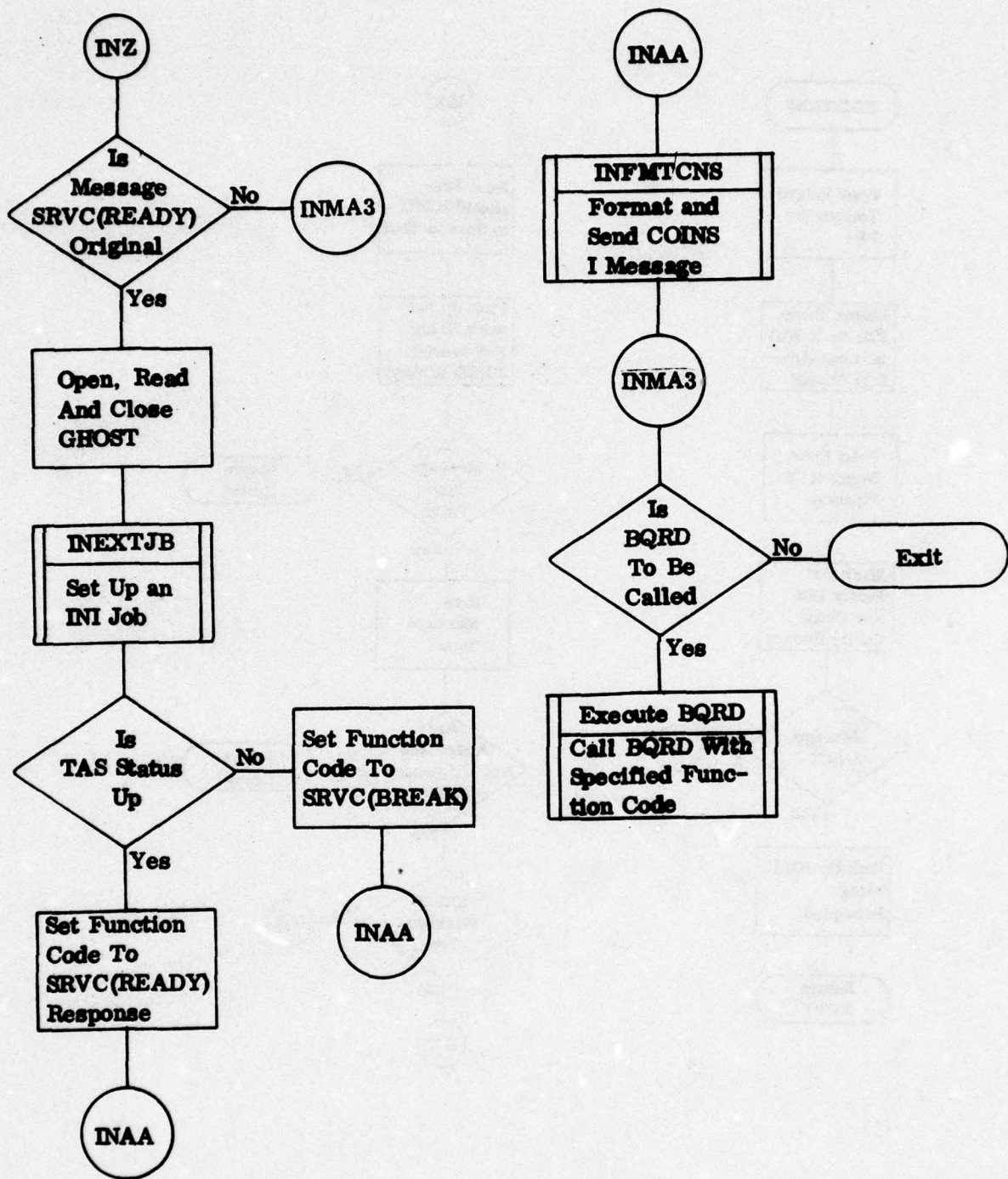


Figure 26. INI Process Data Flow (Sheet 10 of 18)

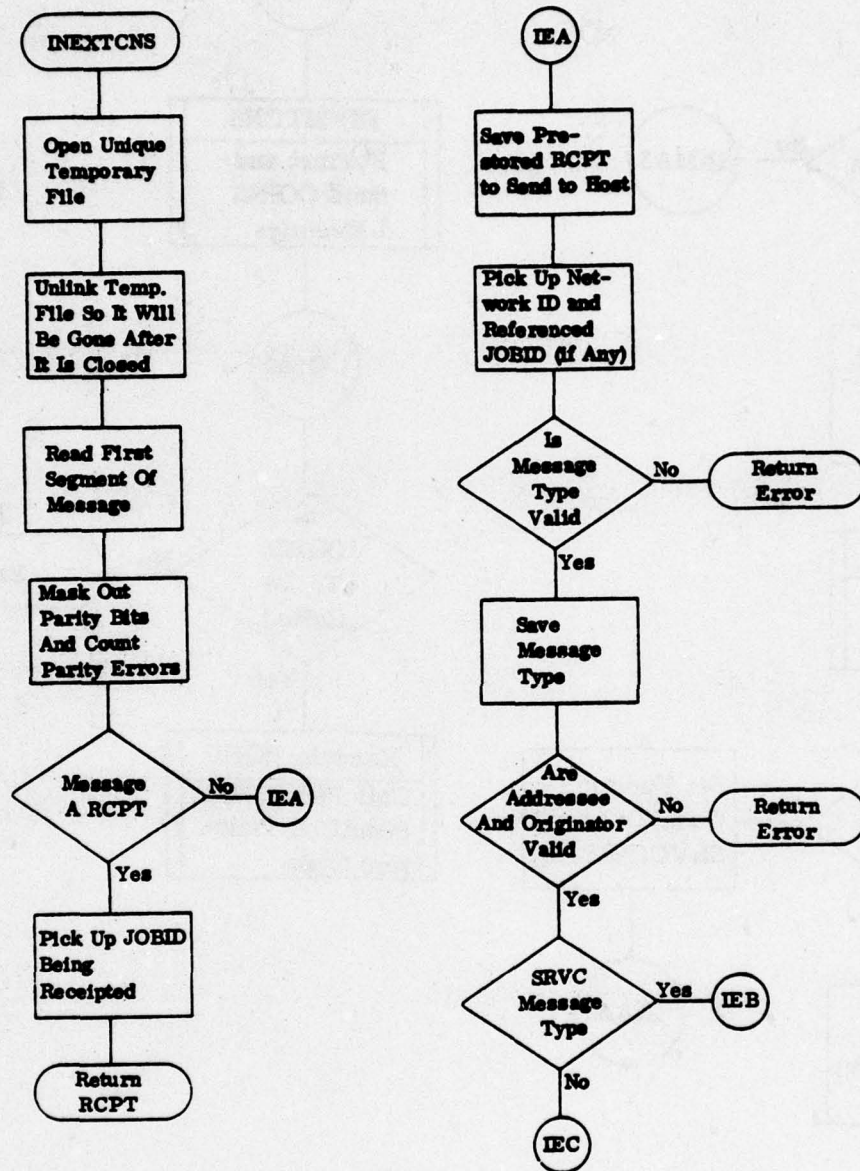


Figure 26. INI Process Data Flow (Sheet 11 of 18)

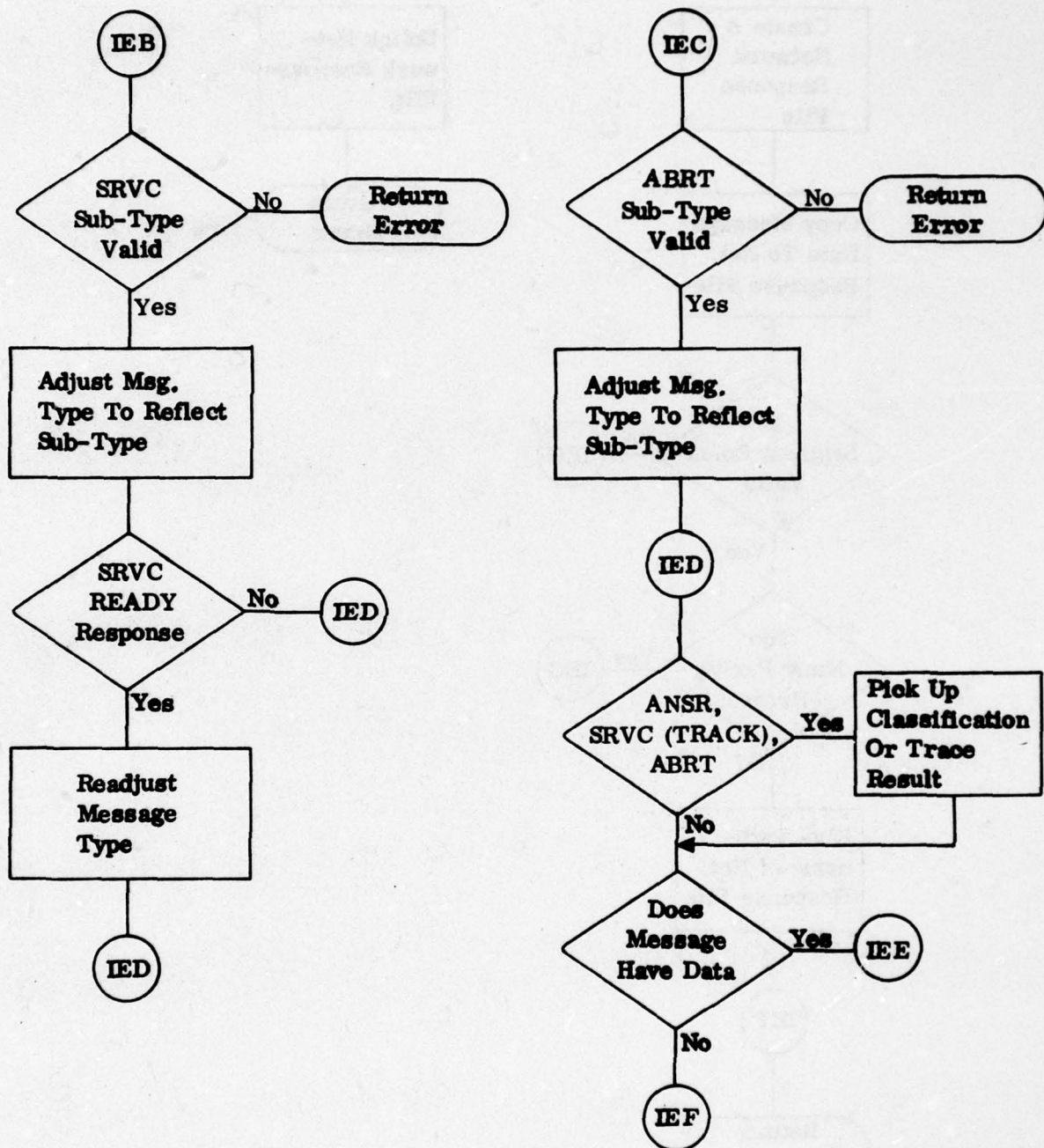


Figure 26. INI Process Data Flow (Sheet 12 of 18)

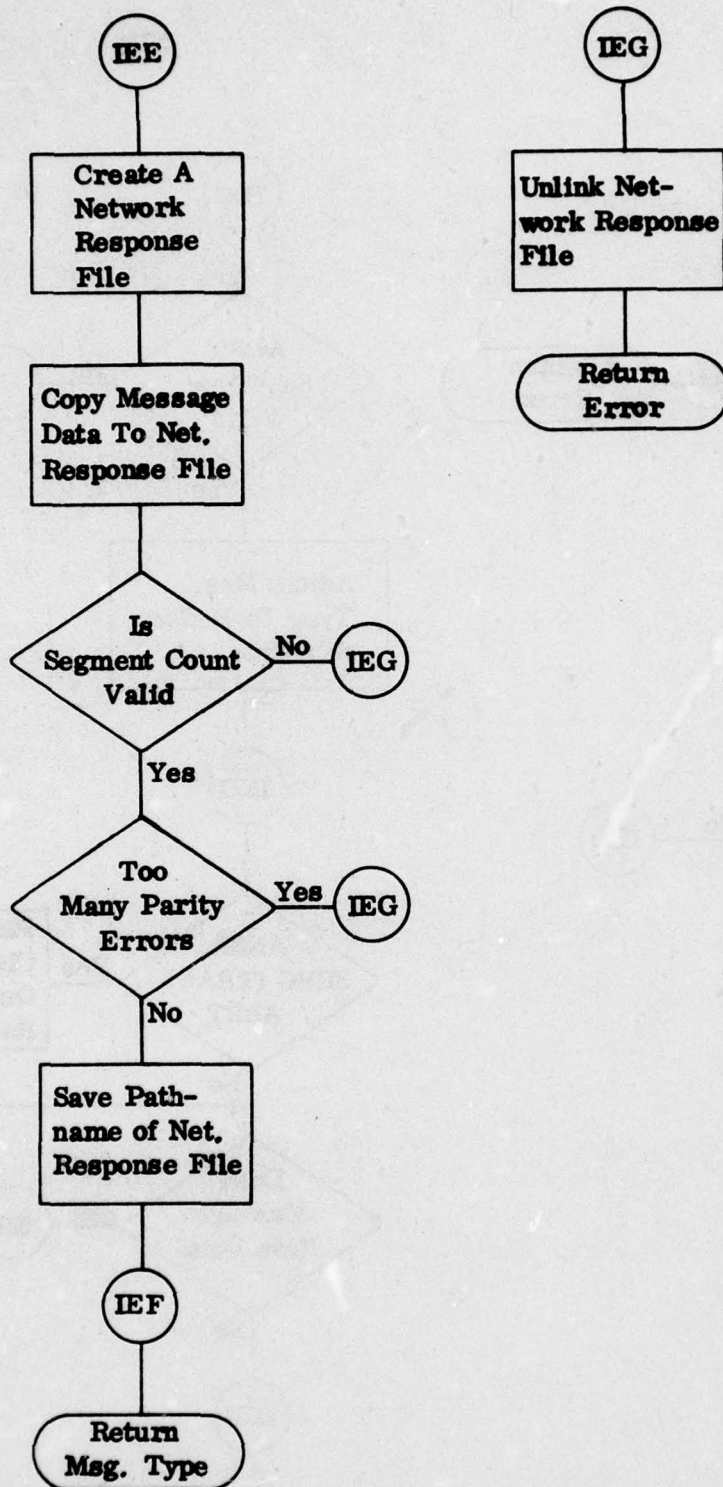


Figure 26. INI Process Data Flow (Sheet 13 of 18)

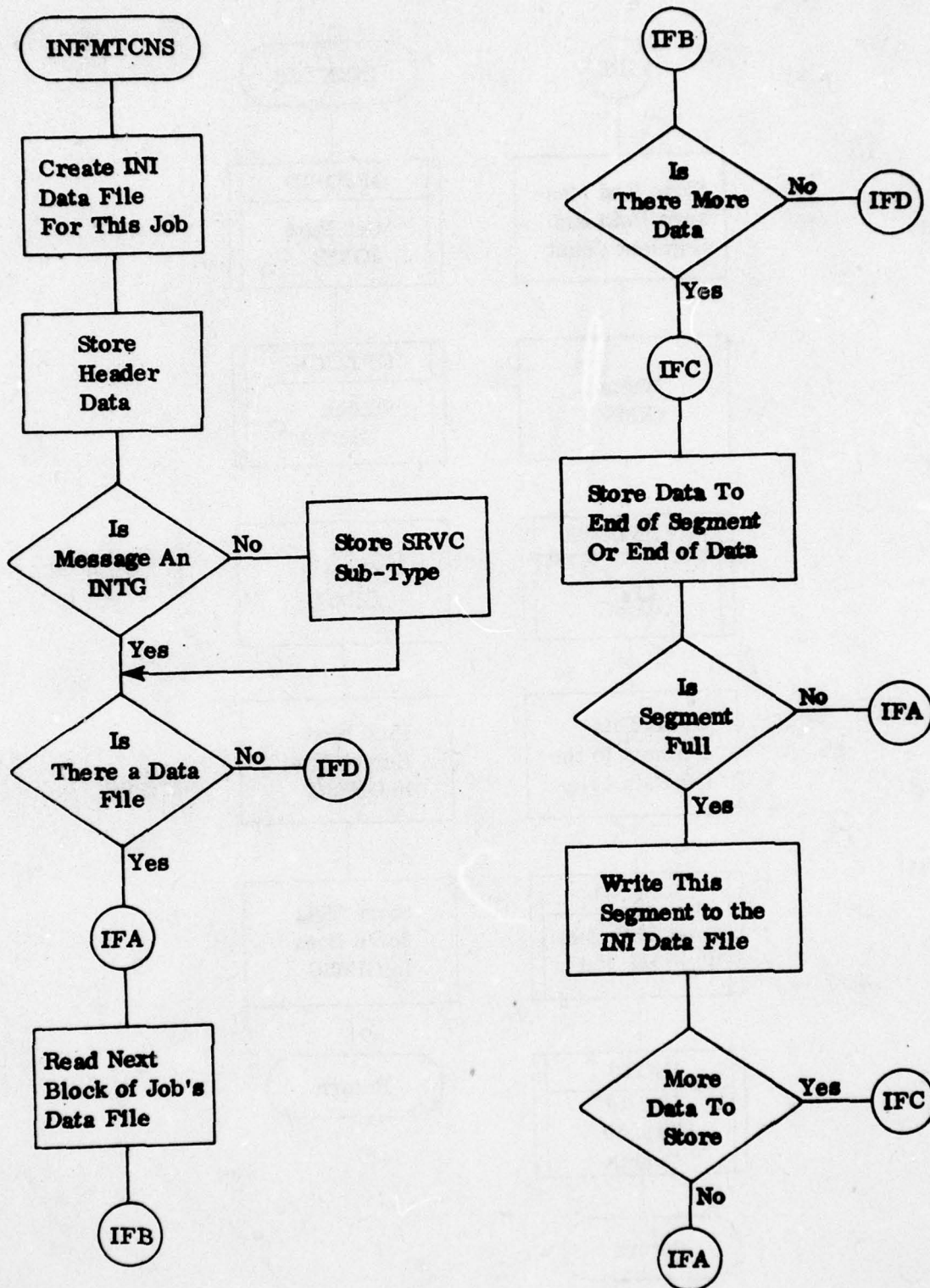


Figure 26. INI Process Data Flow (Sheet 14 of 18)

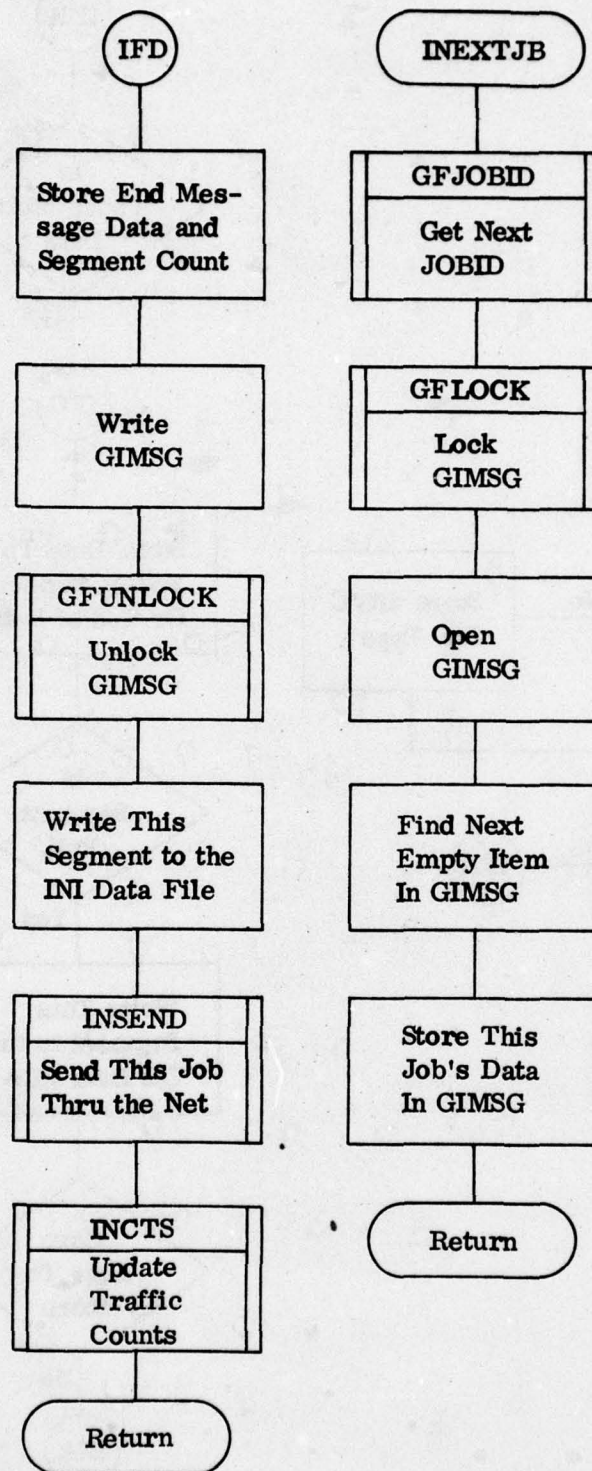


Figure 26. INI Process Data Flow (Sheet 15 of 18)

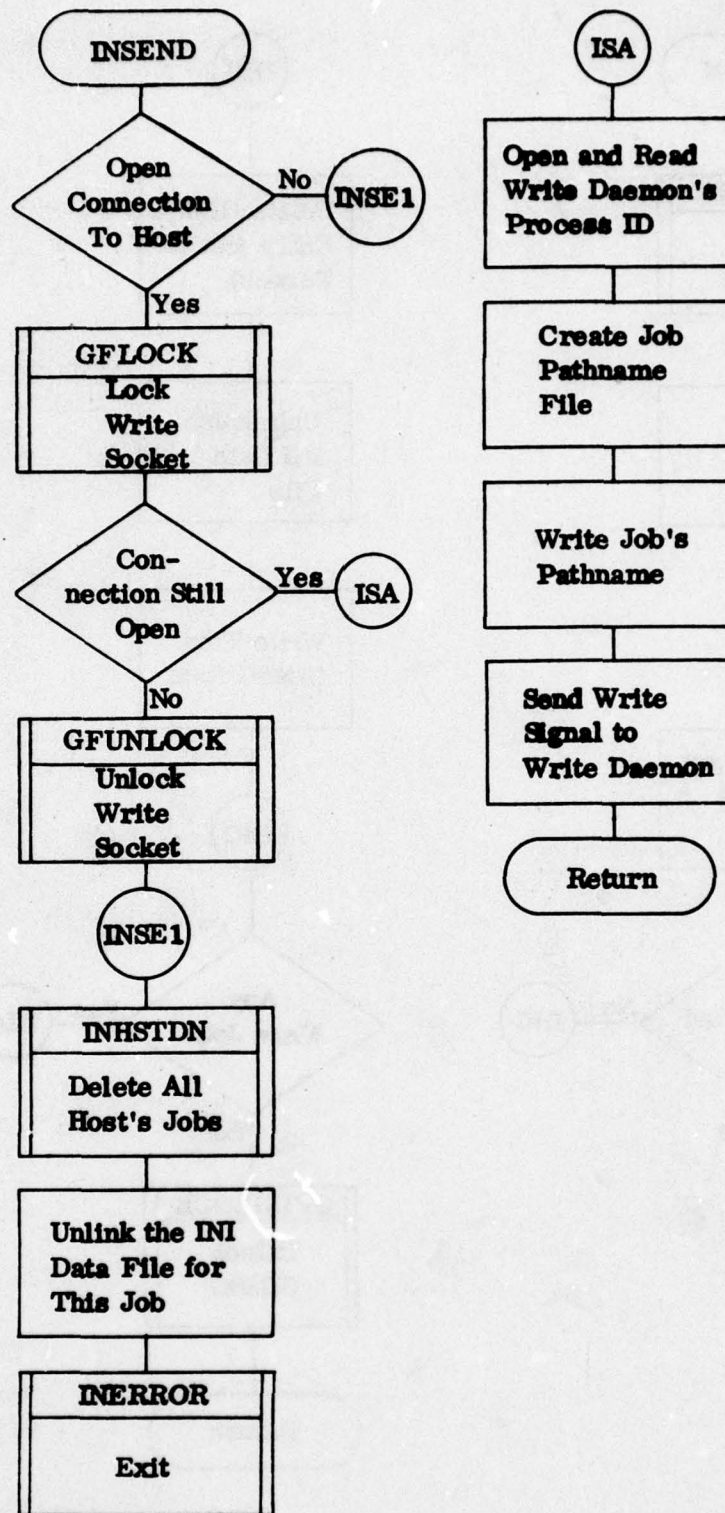


Figure 26. INI Process Data Flow (Sheet 16 of 18)

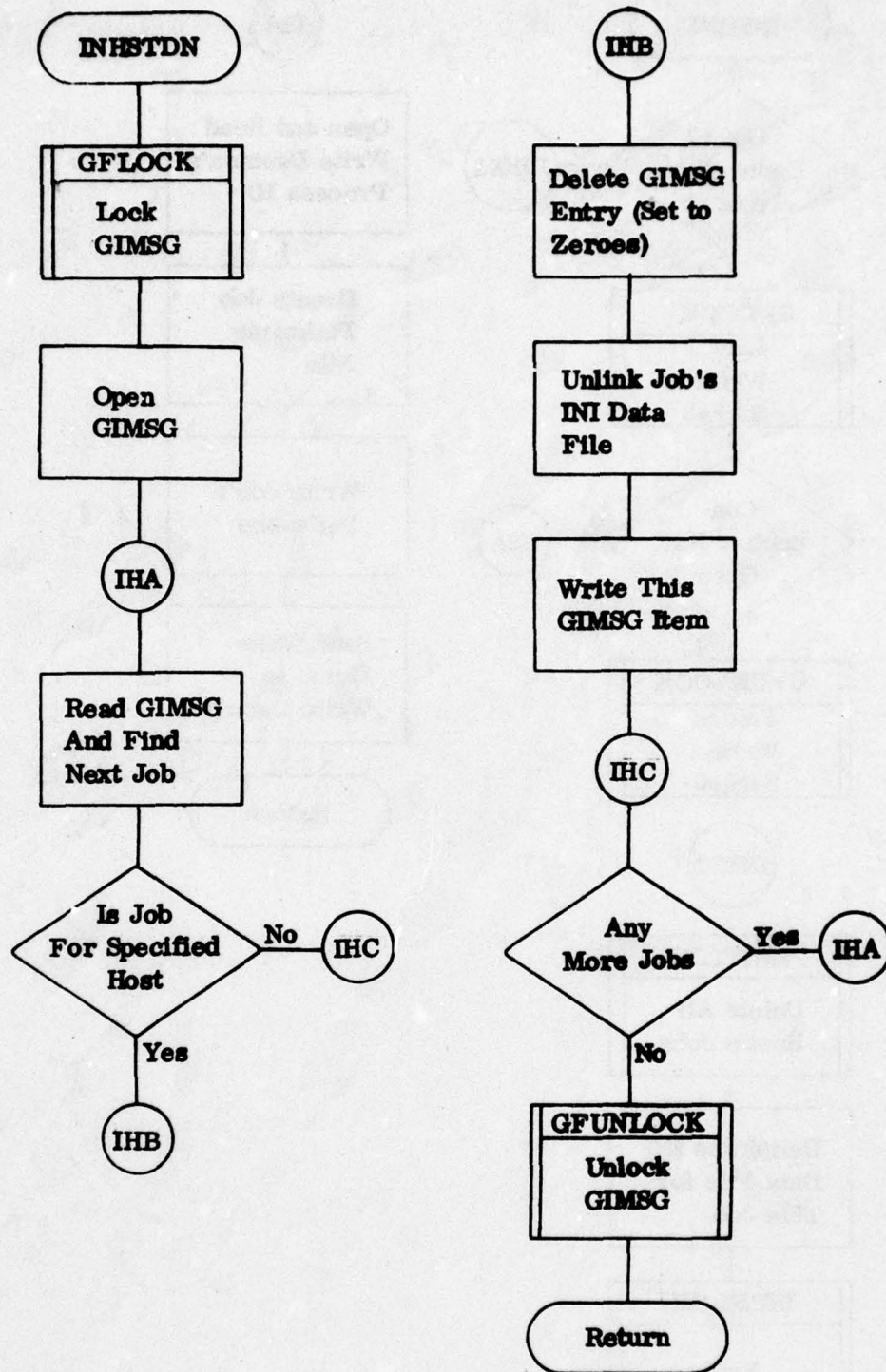


Figure 26. INI Process Data Flow (Sheet 17 of 18)

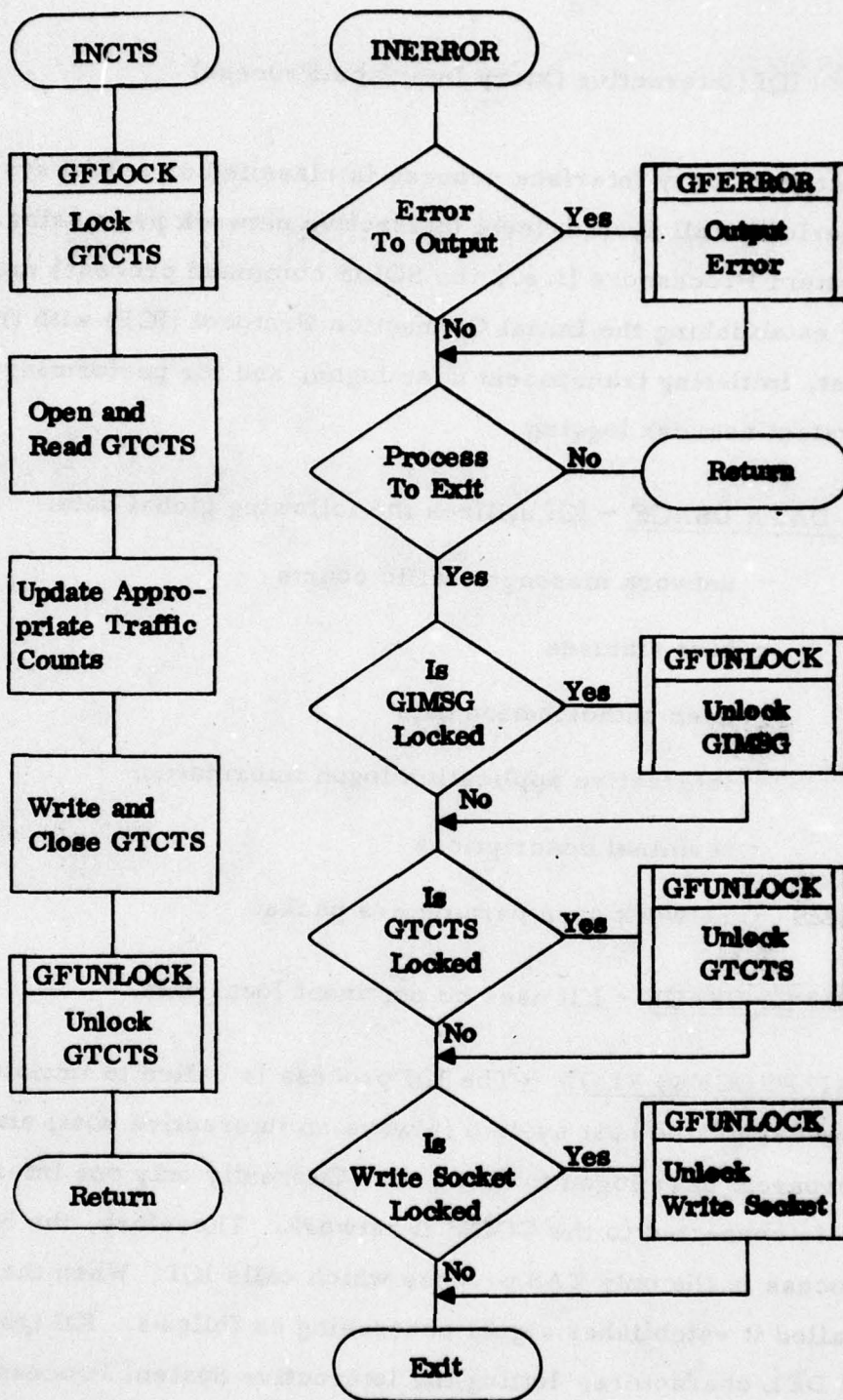


Figure 26. INI Process Data Flow (Sheet 18 of 18)

IQI (Interactive Query Interface Process)

The Interactive Query Interface process is classified as a TAS system process and performs all system level interactive network processing. All Interactive System Processors (i. e., the SOLIS command process) must utilize IQI for establishing the Initial Connection Protocol (ICP) with the designated host, initiating transparent user logon, and for performing all interactive system network logging.

GLOBAL DATA USAGE - IQI utilizes the following global data:

- GTCTS - network message traffic counts
- GHOST - host statuses
- GAUTH - user authorization data
- GILOG - interactive application logon information
- GTERM - terminal descriptions
- GOPARAMS - network open parameters packet

LOCAL DATA USAGE - IQI uses no pertinent local data.

GENERAL PROCESS FLOW - The IQI process is called to initiate an ICP with some designated host system (always an interactive host) and to perform transparent user logon to that host. Currently only one interactive host, SOLIS, is connected to the COINS II network. Therefore, the SOLIS command process is the only TAS process which calls IQI. When the IQI process is called it establishes signal processing as follows. IQI ignores user entered DEL characters, letting the Interactive System Process

(SOLIS command process) trap the DEL and convey the situation to IQI via another signal type. IQI then enables the signal which processes network read error indication signals coming from its network read counterpart, IQIR. Following signal initialization IQI initiates an ICP to the designated host, trying two times before designating the host as down and terminating. Upon successful completion of the ICP, IQI initiates the special host-dependent user logon sequence for that user. If the logon sequence is successful, IQI indicates this condition to the calling process (SOLIS command process), spawns the IQIR read process, and then goes into a tight loop where it alternately reads data from the pipe and writes it out to the network. Data read from the pipe is from the SOLIS command process and represents data coming from the user's keyboard. (See figure 27 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

IQINTER (IQI Interactive Processor) - IQINTER is the primary function in IQI. It is responsible for telling the user that an ICP is being attempted and then initiating that ICP. IQINTER will attempt the connection twice before indicating that the host is not available. The function IQHOST is called to update the GHOST system table, if applicable, indicating the status of the connection. GHOST is updated if the connection result indicates a state contrary to the current state now represented in GHOST. Following successful connection to the host, IQINTER calls the appropriate host-dependent user logon function, which for the SOLIS host is IQSOLON. The logon function returns with either a successful logon indicator (a zero) or an unsuccessful indicator (a negative two). In either case the network traffic counts structure, GTCTS, is updated to reflect the interactive session. Since GTCTS is update-sensitive, it must be locked out via GFLOCK during

this sequence. If the user logon sequence was not successful, IQINTER calls the COINI maintenance function IQLOG to initiate a COINI record representing the unsuccessful attempt, and then notifies the calling process (SOLIS command process) of the unsuccessful attempt by sending a negative two across the pipe. Then IQINTER exits for the IQI process. If the user logon sequence is successful then IQINTER calls IQLOG to initiate a COINI record (indicating successful interactive session logon) and then indicates this successful sequence by writing a zero across the pipe to the calling process. Now IQINTER performs a fork/execute sequence to spawn the read counterpart of IQI, IQIR, passing it the network file descriptor, its (IQI's) process-id for signal sequences, and the write-file descriptor of the pipe. Finally, IQINTER enters a loop sequence where it reads on the pipe awaiting user entered terminal data via the calling process (i.e., SOLIS command process). For each set of data read from the pipe, IQINTER writes it out to the network to the interactive host. It also accumulates the number of characters for this sequence. This data is required for the completion of the COINI record representing this interactive session. This read-pipe/write-network sequence continues indefinitely until one of three events occurs:

- a. the user terminates the interactive session from the TAS side by entering a DEL character and indicating that he wishes to quit. This software interrupt is trapped by the calling process which in turn sends a signal to IQI indicating this condition.
- b. IQINTER encounters a network write error. For this situation the function IQTERM is called to terminate the interactive session.
- c. the IQI read process, IQIR, encounters a network read error and sends a terminating signal to IQI. Refer to function IQDONE.

Typically in case c, the user has terminated his SOLIS session by entering a period ('.') as a SOLIS command hence causing SOLIS to close the connection from its side. This is detected as a network 'read error' by IQIR.

IQHOST (Host Status Processor) – IQHOST is responsible for maintaining the TAS Host status data structure (GHOST) for all interactive hosts sessions. IQHOST is sent an input parameter which determines whether the ICP to the designated host was successful. This input parameter is compared to the current state of the host as specified in GHOST. If the current state differs from the input state, then GHOST is updated to reflect this new condition and a special systems' message indicating this condition is sent to the TASTMASTER at the TAS operator's console via GFERROR. Since IQHOST may modify GHOST, it must be locked out via GFLOCK during this process.

IQSOLNR (Special SOLIS Network Read Processor) – IQSOLNR is a host-specified function in IQI and is responsible for reading from the SOLIS host until an end-of-text (ETX) character is encountered. This function is used for special user SOLIS logon sequences. All SOLIS output, usually in the form of formatted menu-screens, is terminated with an ETX character. IQSOLNR will call IQPRENE, which exits if a network read error is encountered, or returns a negative status if the maximum number of characters was read from the SOLIS host and an ETX character was not encountered. For normal conditions, IQSOLNR returns the number of characters read in from the SOLIS host including the ETX character. Once user authentication has been completed, IQI will relegate network reading to the IQIR process.

IQSOLON (SOLIS User Logon Processor) – IQSOLON is a SOLIS-specific function in IQI which performs all host interaction required in order to successfully log a user on to SOLIS. IQSOLON does this by sequencing through a set of predetermined SOLIS screens which constitute a user's authentication to that system. Prior to user authentication, IQSOLON must identify the terminal type (TTY MOD-40), terminal location (special SOLIS trigraph), host network identifier, and a special terminal logical line number. This data is SOLIS peculiar and therefore is embedded in the IQI process. Upon successful terminal/host identification, IQSOLON then proceeds to authenticate the designated user. First, IQSOLON must read in the SOLIS generated option-screen which it then answers with a 'canned' screen specifying that this is an 'operational' user who desires to retrieve from the PRODUCT data base (i.e., RP option). Then IQSOLON reads the next screen in the logon sequence, the authenticate-screen. Now IQSOLON must dynamically build the response to this screen by inserting the user's logon data contained in GILOG; i.e., social security number, name, and a special three-character password. This fabricated screen is then written to the SOLIS host via the network. Finally, IQSOLON reads in the authenticate-response insuring that the user does indeed have access to the SOLIS host. Two possible screens can be received from SOLIS specifying successful authentication. One, a 'scheduling your request' screen indicating successful authentication, and two, 'xyz is now your new password', indicating successful authentication and also specifying the user's new SOLIS password. For the latter case, IQSOLON must update the user's logon data file (GILOG) with this new password. In either case, this SOLIS screen is output to the user as his first screen from SOLIS. If the SOLIS host does not indicate successful authentication of the user, then

IQSOLON returns with a status of negative two. A zero return value indicates a successful status.

IQLOG (IQI Logging Processor) – This function is responsible for establishing and terminating the COINI interactive log record. IQLOG is called after user authentication and it will establish either a successful or unsuccessful logon attempt. At this time a system JOBID is established by calling GFJOBID. Then the appropriate TAS Logging Process is called to satisfy the logging event. The actual contents of the TLP process parameters are built by the other functions prior to calling IQLOG. IQLOG also waits for the TLP process to exit prior to returning.

IQTERM (IQI Terminate Signal Processor) – IQTERM is called either by the function which traps the terminate signal from the SOLIS command process, or internally by IQINTER. In either case IQTERM ensures that no locks are set (either GHOST or GTCTS) and calls GFUNLOCK to unlock the files if locked.

Then, prior to signalling the IQI read process, IQIR, to terminate, IQTERM determines whether the IQI sequence has reached this point. It is possible for the user to terminate an interactive transaction prior to completing the ICP or the user authentication. If IQIR is not present, then either the ICP or user authentication has not been completed and IQTERM does not send a terminating signal to IQIR (which doesn't exist). Under this condition IQTERM determines whether a COINI record has been established and calls IQLOG to terminate the record if it has. In either case IQTERM then exits for the IQI process. If IQIR does exist, i.e., the ICP and user authentication

(logon) has been established, IQTERM must send a terminate signal to IQIR and then sleep until IQIR has terminated and written out the number of characters read from the network during this interactive session. A signal from IQIR will wake the IQI process for further processing of this termination cycle.

IQDONE (Interactive Session Completed) – IQDONE is called by the function which traps the completion signal from IQIR. The temporary file containing the number of characters read during this interactive transaction (created by IQIR) is read in by IQDONE and the char-count is added to the char-count already accumulated by IQI. Then the temporary file is unlinked. The COINI record is terminated by calling IQLOG with a terminate parameter. IQDONE then checks to see if this is an abnormal termination (via host termination or a network error). If it is, then IQDONE sends a terminate signal to the SOLIS process.

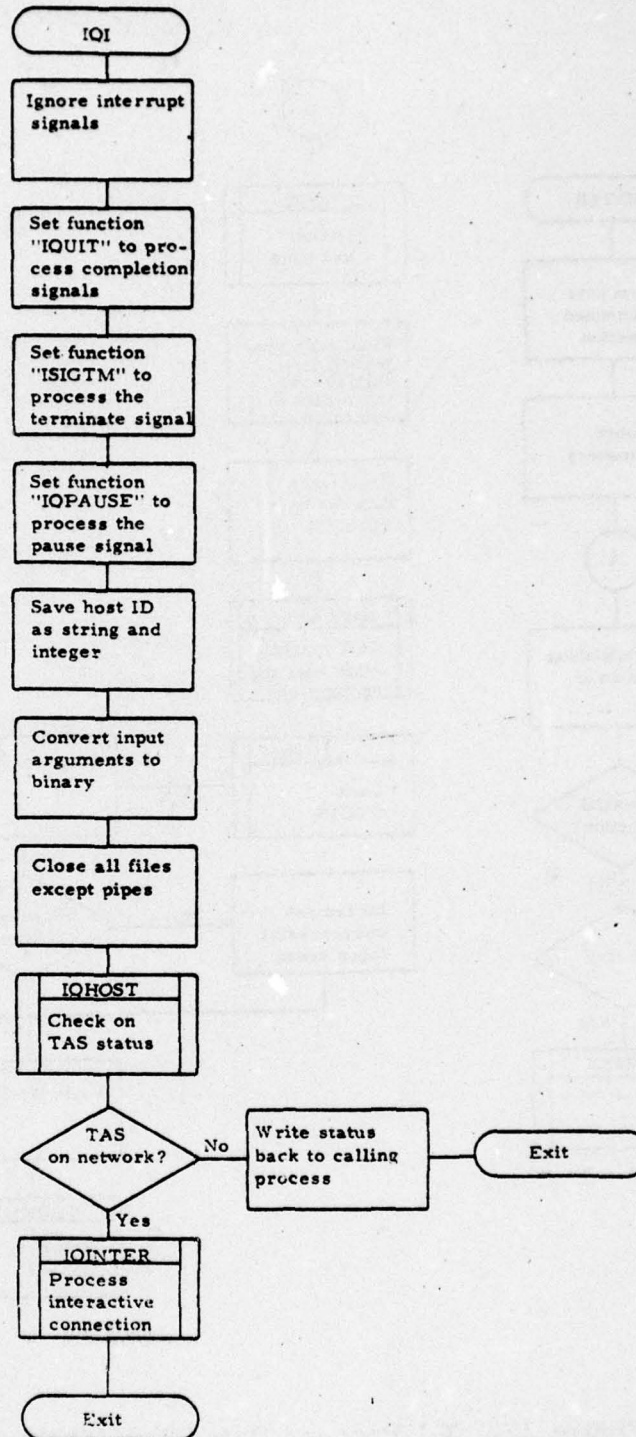


Figure 27. IQI Process Data Flow (Sheet 1 of 8)

AD-A050 967

LOGICON INC SAN DIEGO CALIF

F/G 9/2

TERMINAL ACCESS SYSTEM (TAS) SYSTEM DESIGN SPECIFICATION (U)

JAN 78 R M BARNHART, L R ERICKSON

N00014-76-C-0899

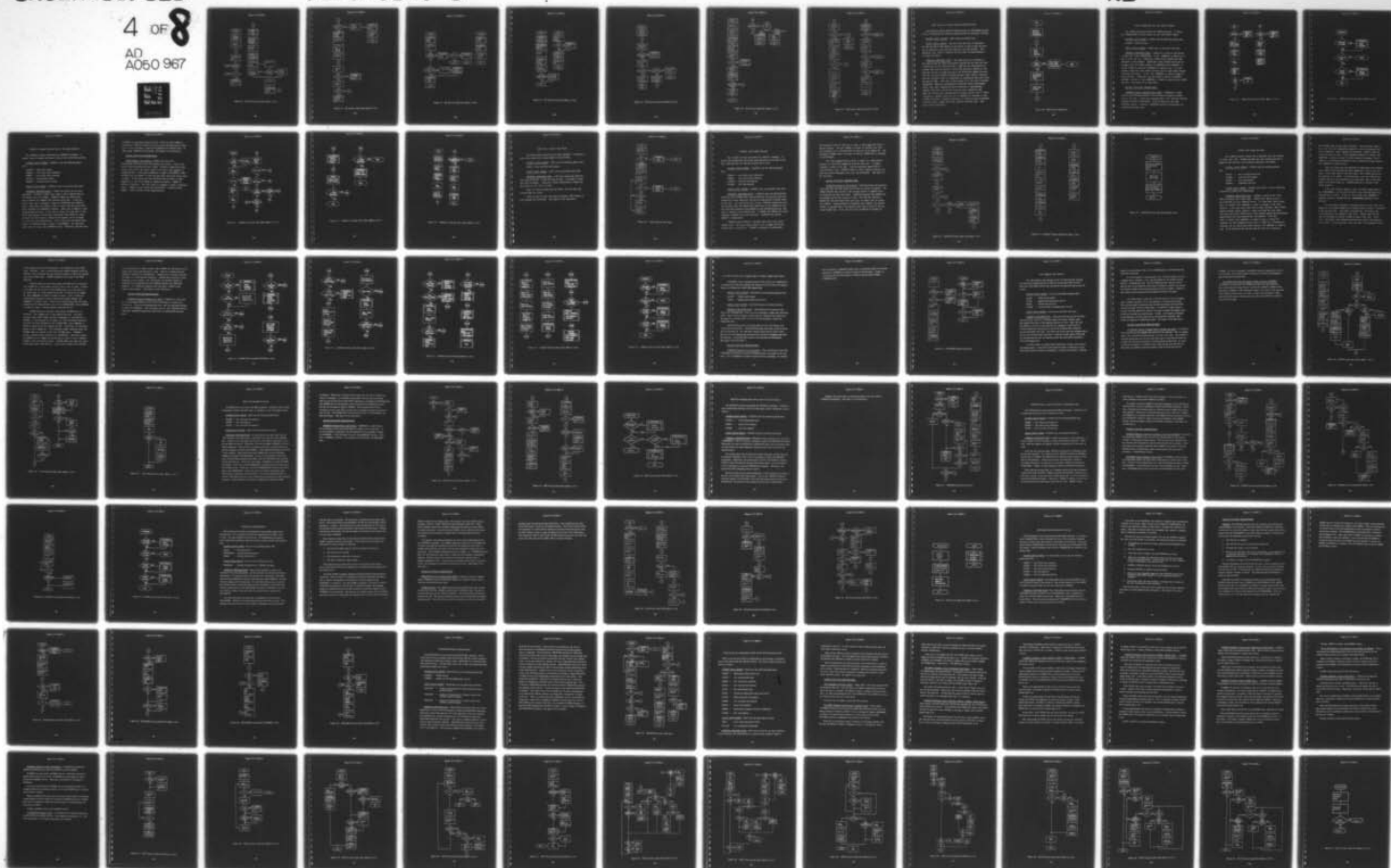
UNCLASSIFIED

76-C-0899-6

NL

4 OF 8

AD
A050 967



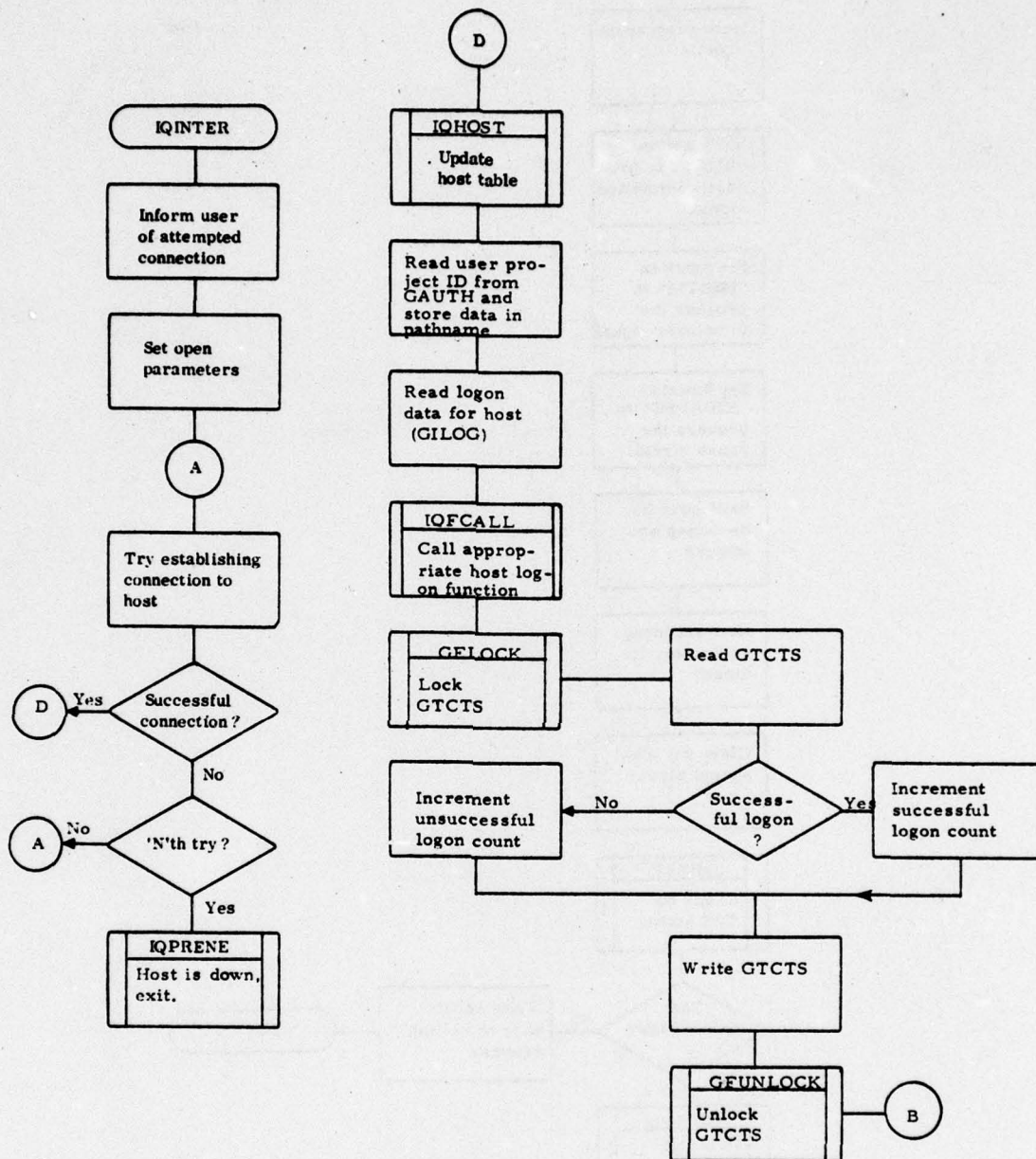


Figure 27. IQI Process Data Flow (Sheet 2 of 8)

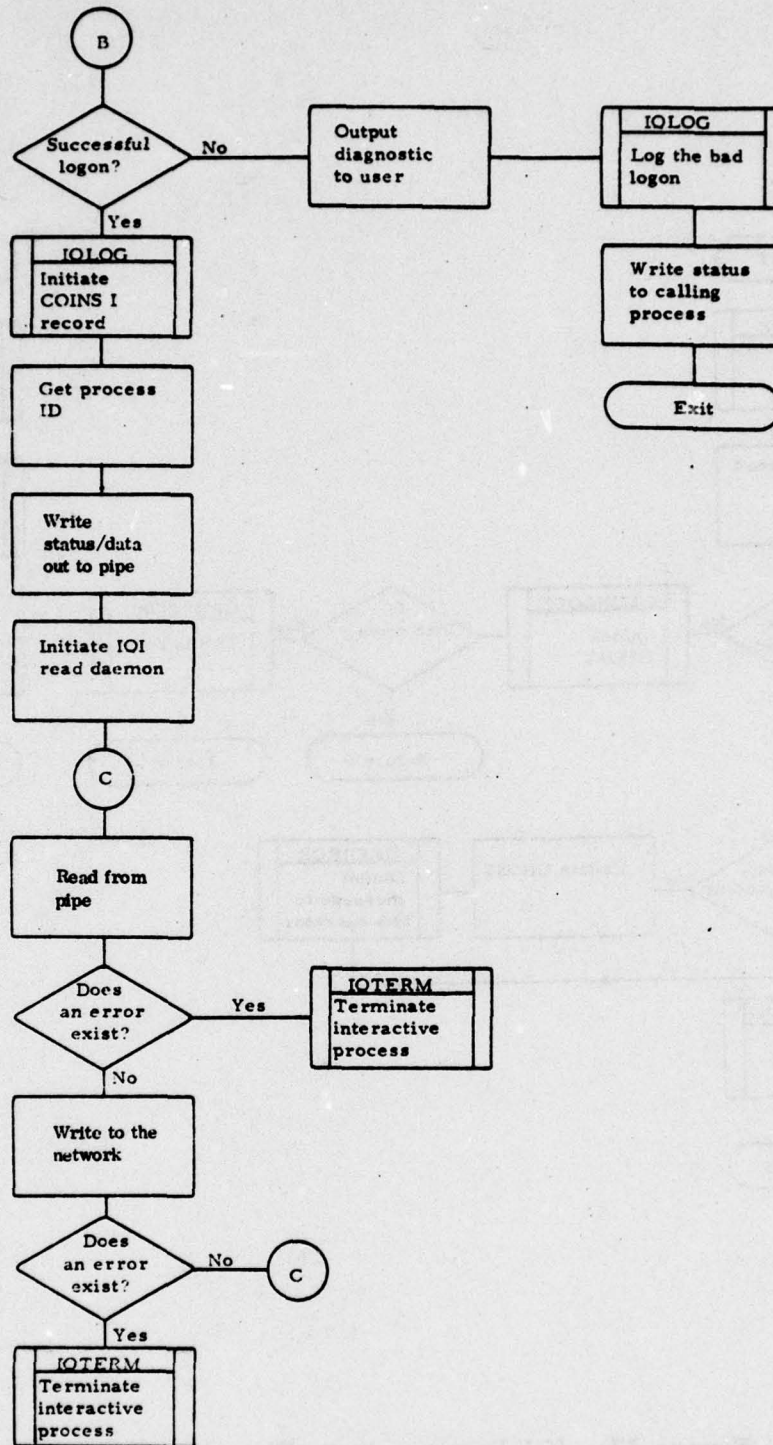


Figure 27. IQI Process Data Flow (Sheet 3 of 8)

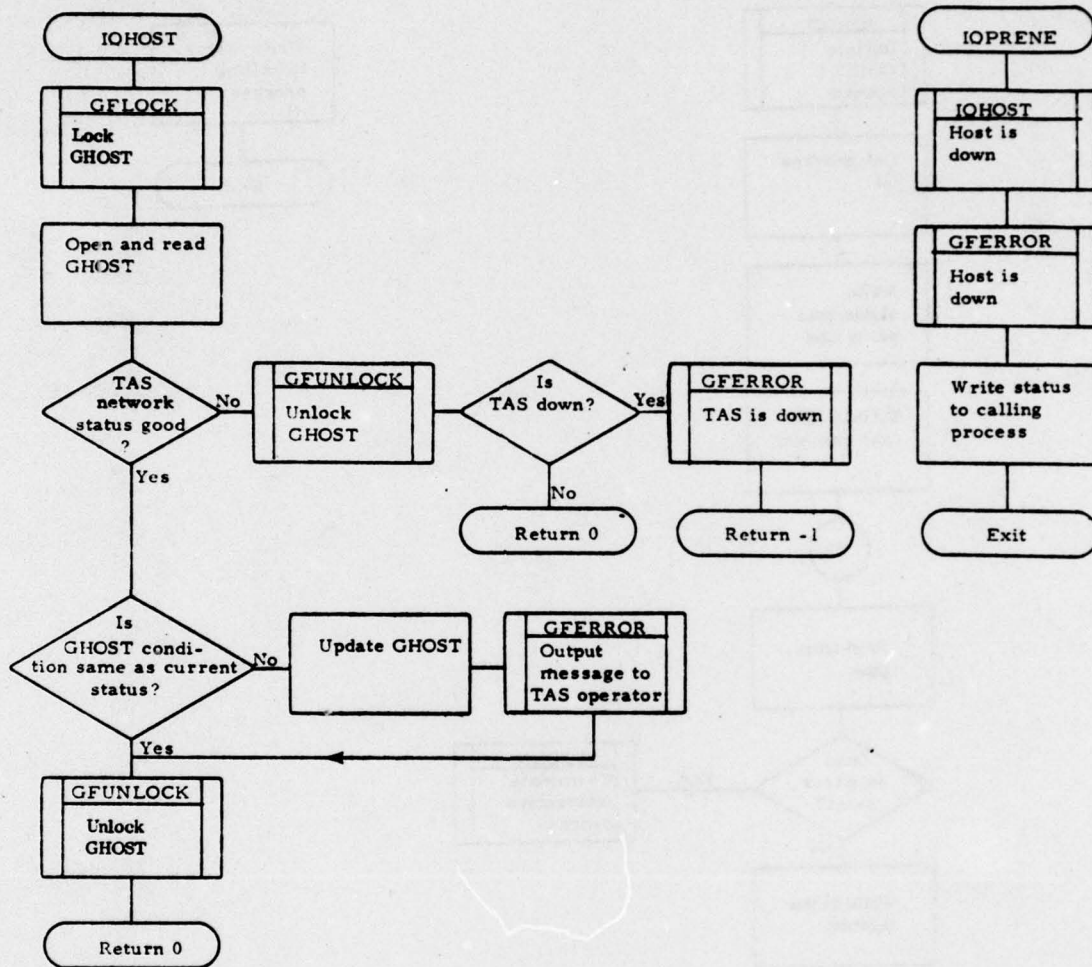


Figure 27. IQI Process Data Flow (Sheet 4 of 8)

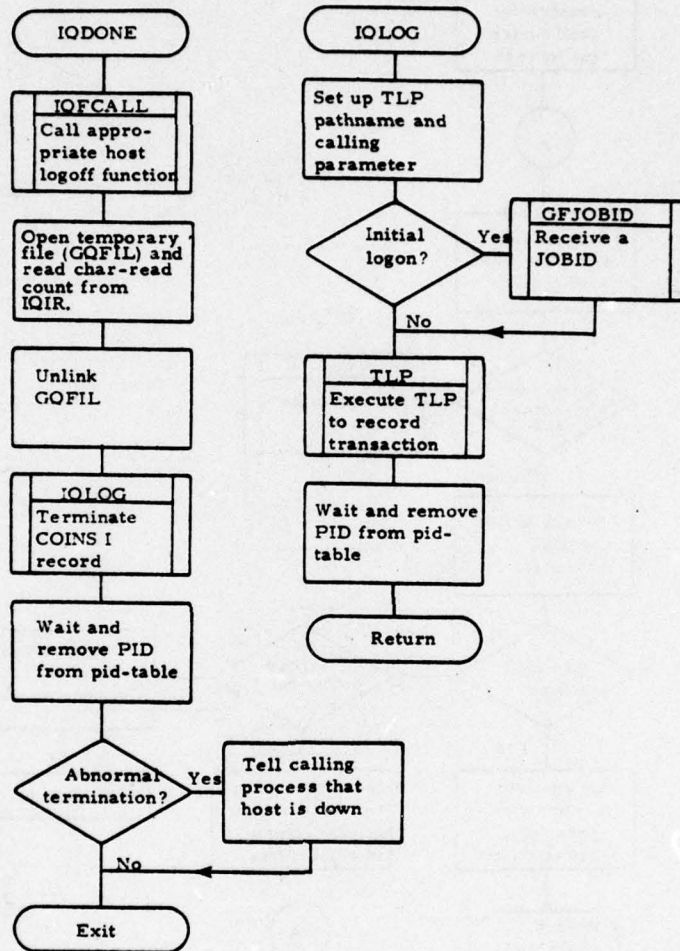


Figure 27. IQI Process Data Flow (Sheet 5 of 8)

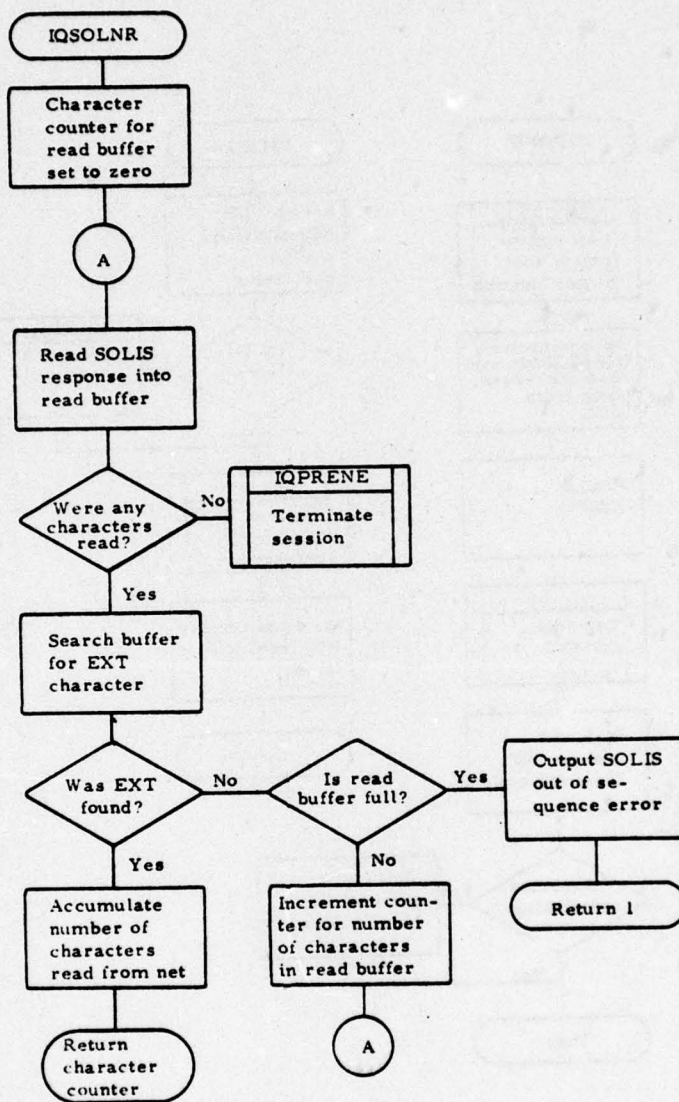


Figure 27. IQI Process Data Flow (Sheet 6 of 8)

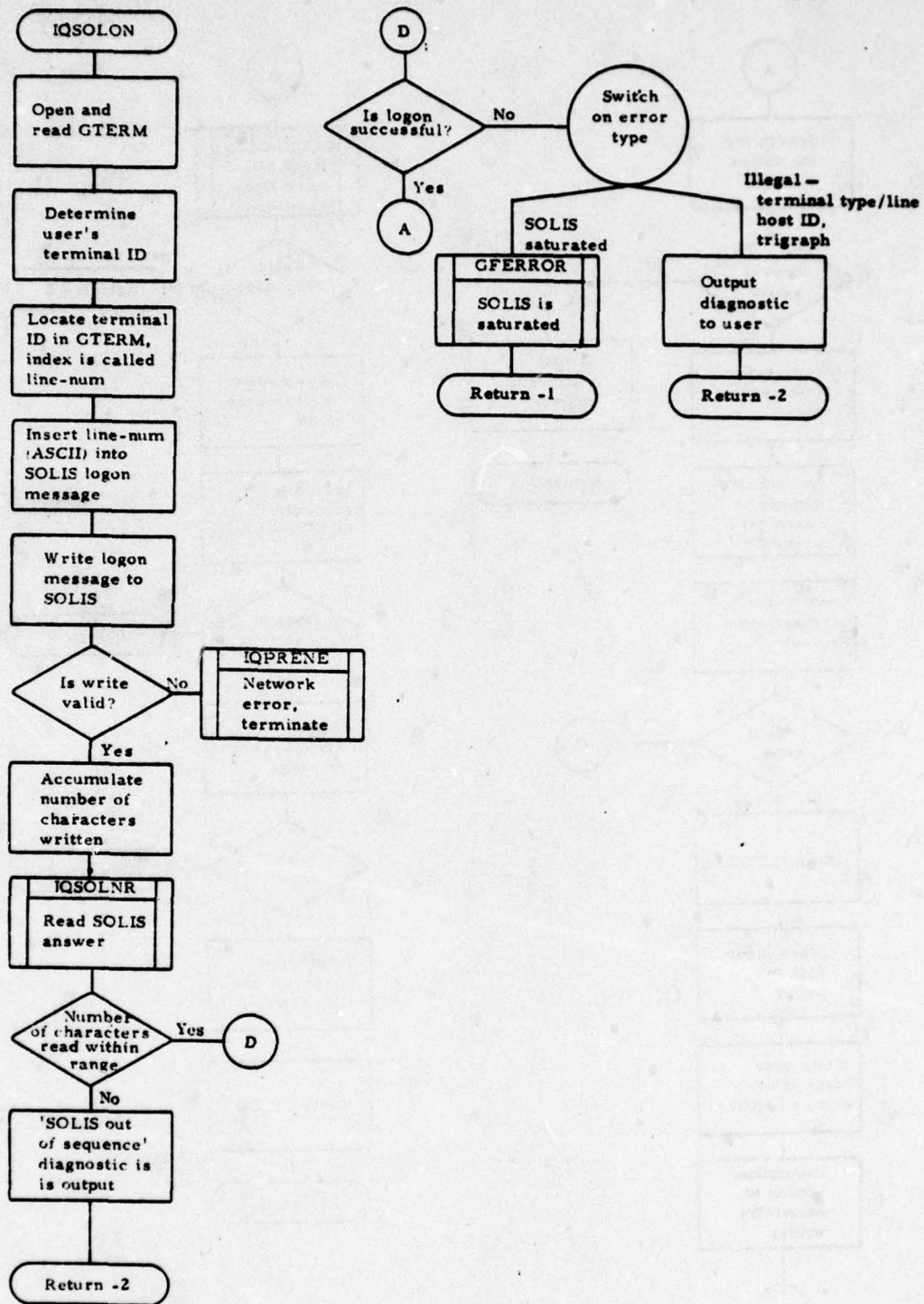


Figure 27. IQI Process Data Flow (Sheet 7 of 8)

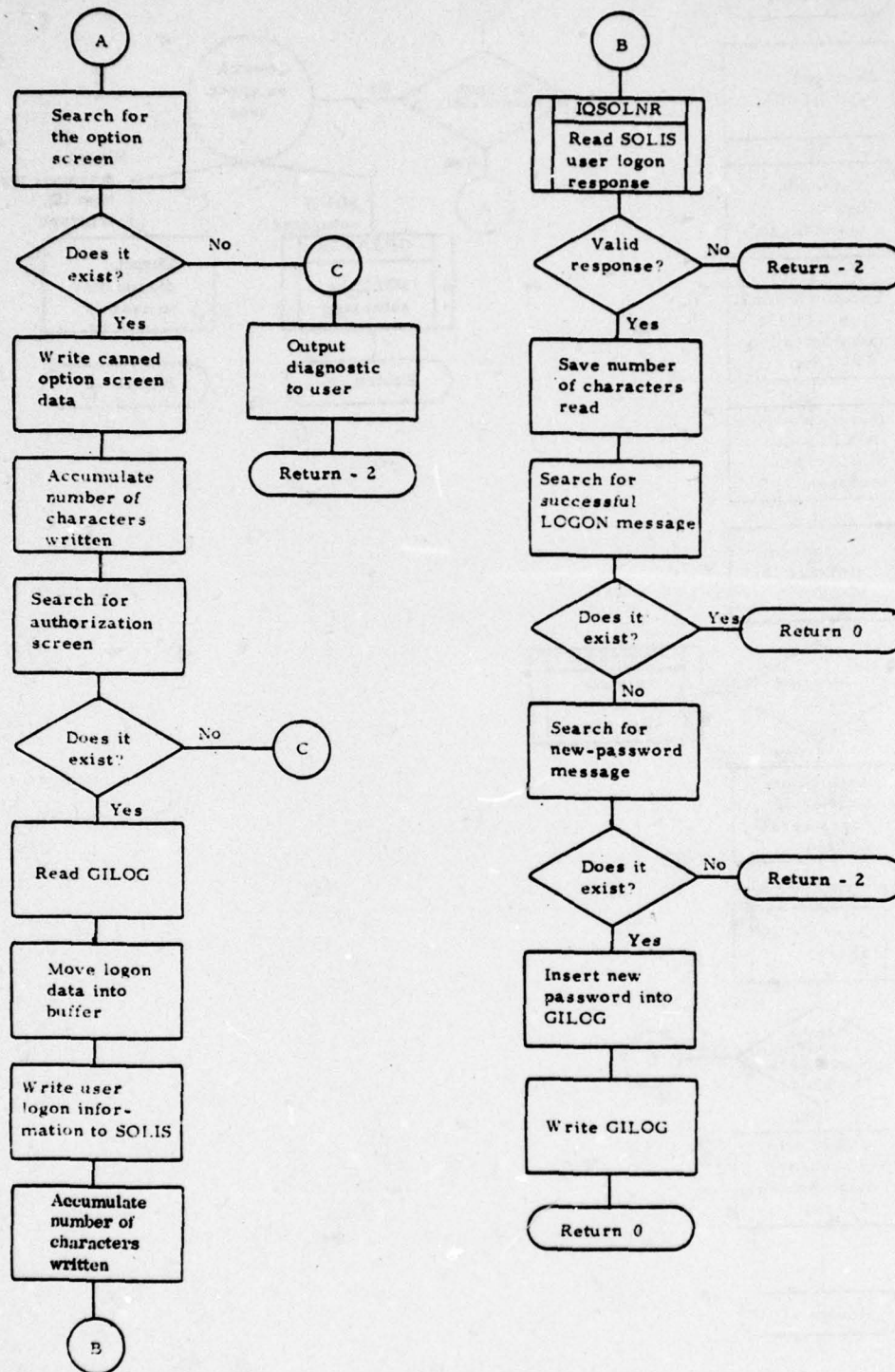


Figure 27. IQI Process Data Flow (Sheet 8 of 8)

IQIR (Interactive Query Interface Read Process)

The Interactive Query Interface Read process is a TAS system process which is responsible for network reading during all interactive sessions.

GLOBAL DATA USAGE - IQIR utilizes no global data.

LOCAL DATA USAGE - The only local data that is significant for this process is the temporary file which is used to pass the number of characters read from the network to the IQI process. The temporary file, which is inserted into the user's current directory, is named 'iqi_temp'.

GENERAL PROCESS FLOW - The IQIR process is spawned by IQI when an ICP has been successfully initiated and the user has been authenticated by the interactive host. IQIR has the responsibility of reading data from the network (via the file descriptor passed to it by IQI) and writing it through the pipe to the Interactive System Process. For the case of the SOLIS command process, there exists a separate process, SOLISW, which is responsible for reading the data sent across the pipe by IQIR. This data is then written out to the user's terminal by SOLISW. IQIR also accumulates the character count of all network reads, and, when signalled by IQI to terminate or encountering a network read error, writes this count out into a temporary file. IQIR will continue this read-network/write-pipe sequence until it either encounters a network read error or is signalled to terminate via IQI. If IQIR encounters a network read error, it writes out the accumulated character count, signals IQI of the condition and then exits. (See figure 28 for data flow.)

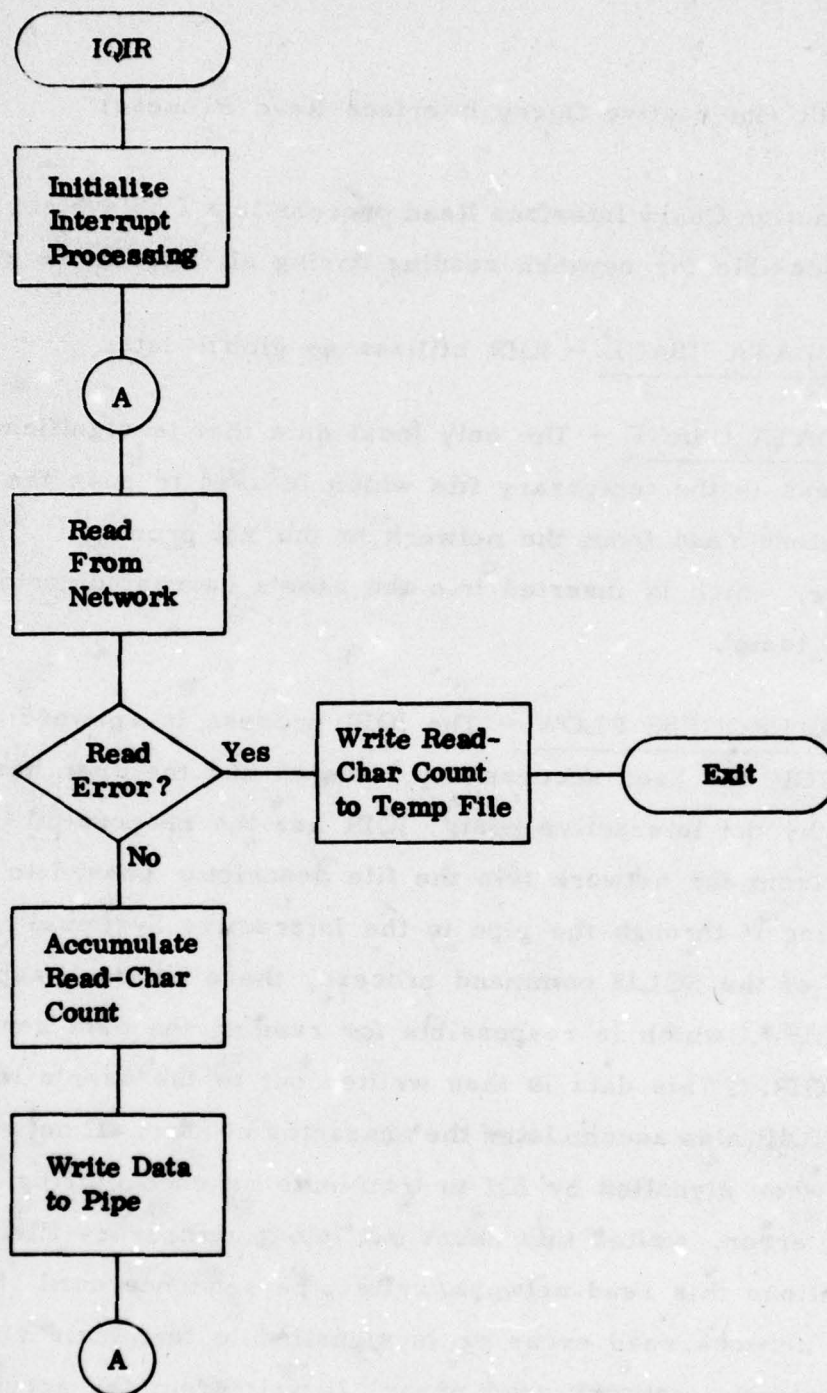


Figure 28. IQIR Process Data Flow

JOBID (Display/set the TAS JOBID Counter)

The JOBID process processes the JOBID command. It allows the TASMASTER to display and/or set the TAS JOBID counter.

GLOBAL DATA USAGE - JOBID uses the following global data:

GJOBID - JOBID Counter

LOCAL DATA USAGE - JOBID uses no pertinent local data.

GENERAL PROCESS FLOW - JOBID first checks to see that the current user is the TASMASTER. If not, JBERROR is called to output an error and exit. Otherwise, JOBID opens GJOBID and calls GFLOCK to lock GJOBID. JOBID then reads GJOBID and prints its contents, the TAS JOBID counter, via JBERROR on the terminal. If no new JOBID is specified, JBERROR is called to exit. If a new JOBID is specified, then JOBID validates that the input value consists of exactly five digits. If not, then JBERROR is called to output an error and exit. Otherwise, the new JOBID is stored in GJOBID, GJOBID is written out and JBERROR is called to output the new JOBID and exit. (See figure 29 for data flow).

MAJOR FUNCTION DESCRIPTIONS

JBERROR (Output Message and/or Exit) - JBERROR is called with an error/message number (or zero) and an exit flag set to either zero or non-zero. If the error/message flag is set, then the specified message is output via GFERROR. If the exit flag is not set, then JBERROR returns. Otherwise, JBERROR unlocks any locked files via GFUNLOCK and then exits.

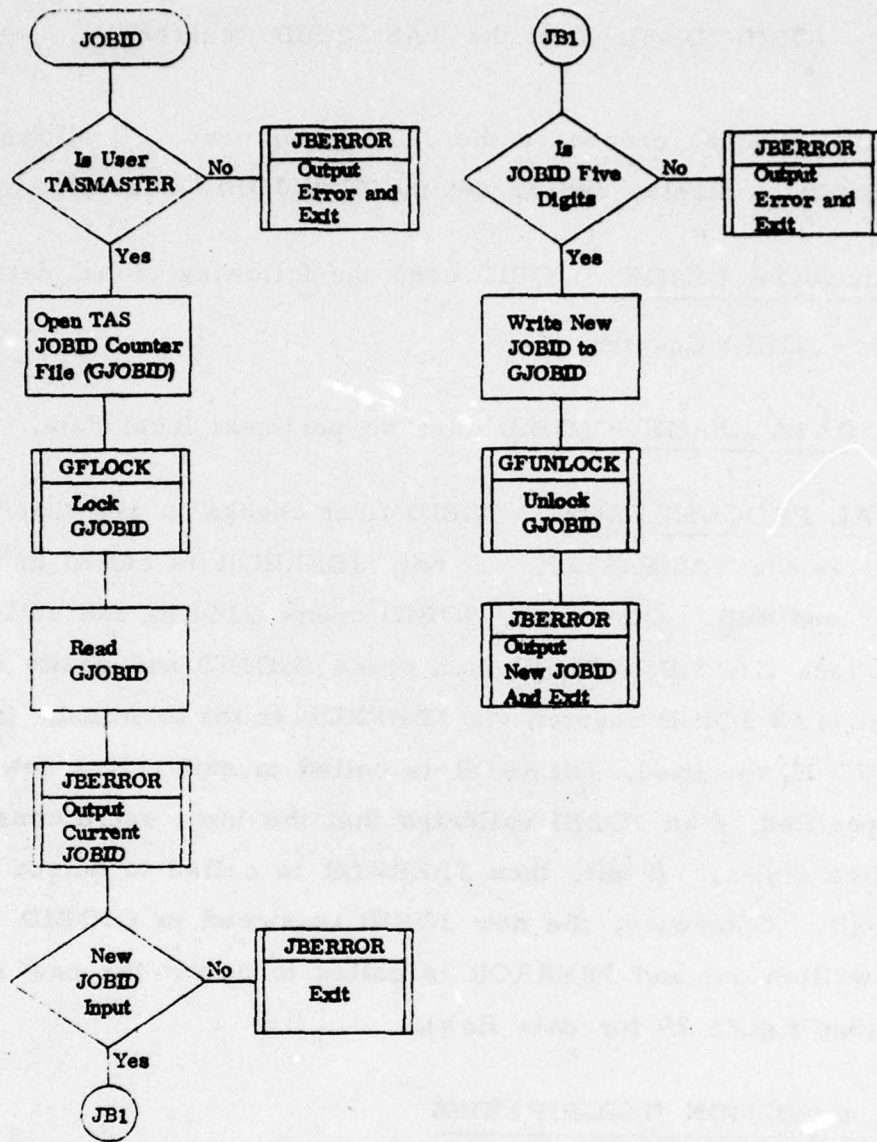


Figure 29. JOBID Process Data Flow (Sheet 1 of 2)

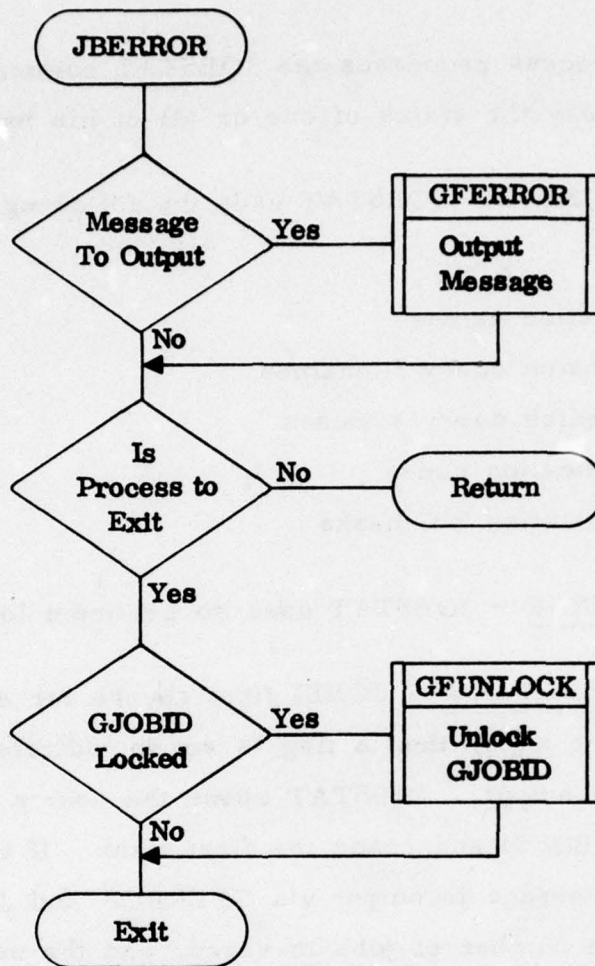


Figure 29. JOBID Process Data Flow (Sheet 2 of 2)

JOBSTAT (Display Status of One or All Batch Queries)

The JOBSTAT process processes the JOBSTAT command. It allows a user to display the status of one or all of his batch queries.

GLOBAL DATA USAGE - JOBSTAT uses the following global data:

- GAPNM - application names
- GBQFN - user batch query filenames
- GBQST - user batch query statuses
- GCLAS - classification codes
- GMASK - classification bit masks

LOCAL DATA USAGE - JOBSTAT uses no pertinent local data.

GENERAL PROCESS FLOW - JOBID first checks for an input parameter. If there is none, then a flag is set to indicate that all job statuses are to be output. JOBSTAT opens the user's batch query statuses file (GBQST) and reads the first item. If the user has no jobs, then a message is output via GFERROR and JOBSTAT exits. Otherwise, the number of jobs is saved, and the user's batch query filenames file (GBQFN) is opened. Each item in GBQST is read and processed as below. If the item is empty, the next item is read. If the item contains a batch query description, then JOUST is called to output the job's status if either all job statuses are to be output or if the job is the specified job. Prior to the first job status being output, a JOBSTAT header is output via GFERROR. If a specified job's status is output, then JOBSTAT exists. Otherwise, the next item

of GBQST is read and processed as above. When the end of GBQST is encountered, JOBSTAT checks to see if a job was specified and not found. If so, an error message is output via GFERROR and JOBSTAT exits. In other cases, JOBSTAT merely exits. (See figure 30 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

JOUST (Output a Job's Status) - JOUST sets up the job's delivered/not delivered status and held/not held status. JOUST then prints a line containing the job's JOBID, referenced application name, status, delivered or not status, held or not status, and response or submission time. A file names subheader is output via GFERROR. Each pertinent item in GBQFN is read and each file's name and access mode is output. If the job has not been delivered and its status is either logged out or lost, then GBQST is updated as follows. GBQST is locked via GFLOCK. The job's status is changed to reflect a delivered status. GBQST is then written and GFUNLOCK is called to unlock GBQST. JOUST returns to the calling function.

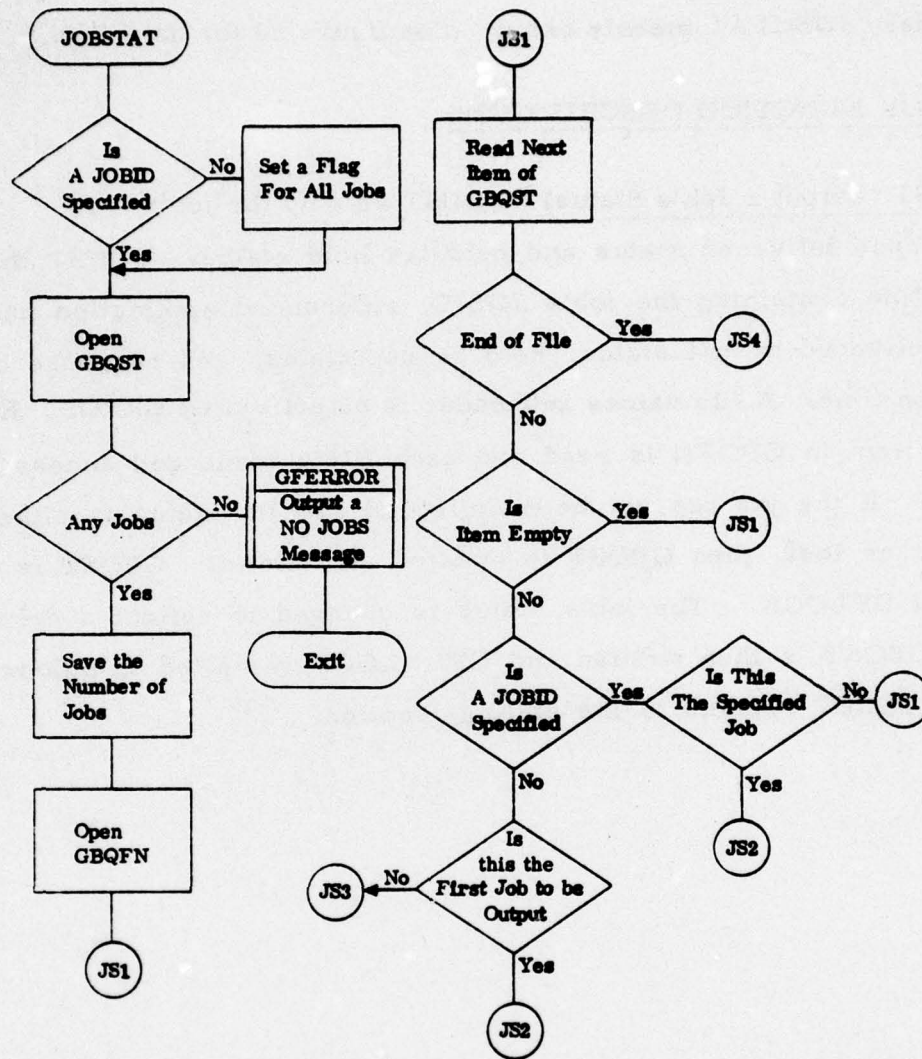


Figure 30. JOBSTAT Process Data Flow (Sheet 1 of 3)

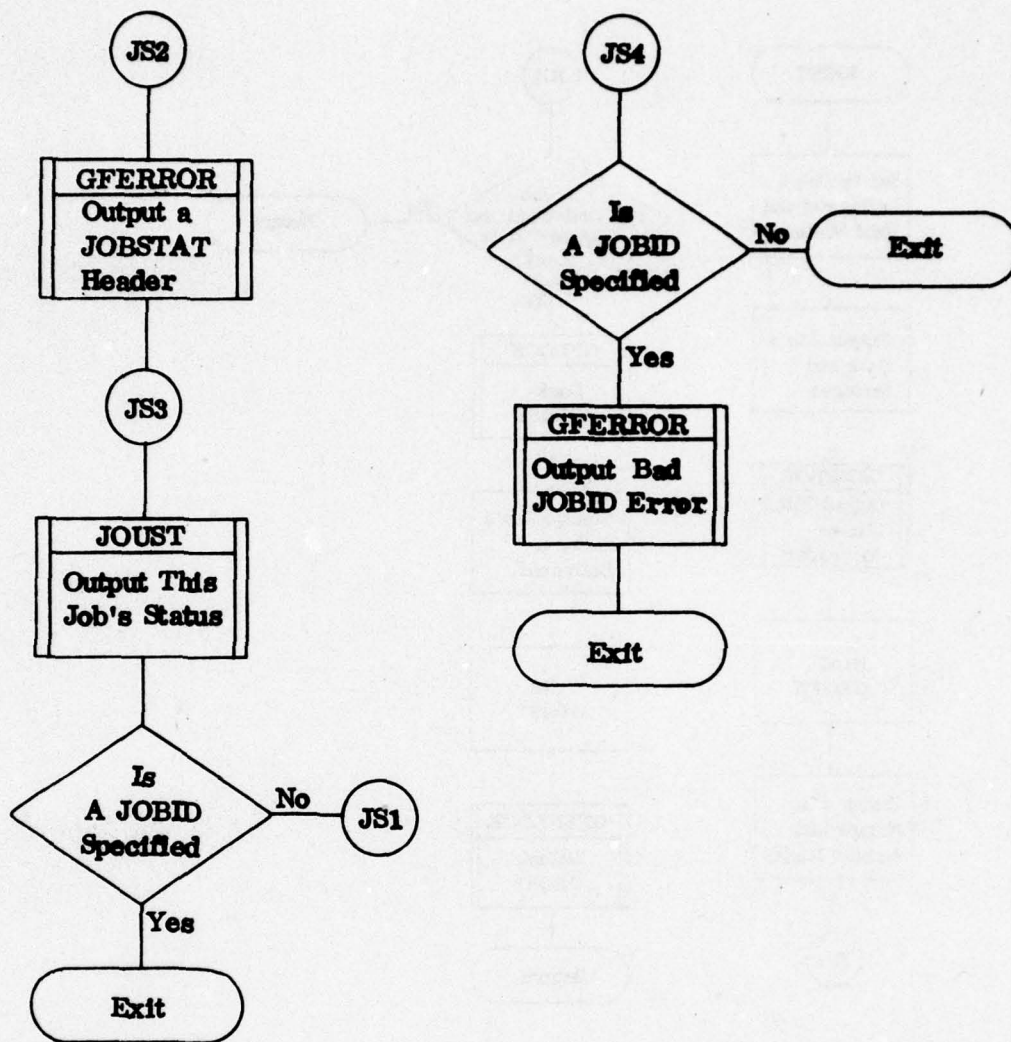


Figure 30. JOBSTAT Process Data Flow (Sheet 2 of 3)

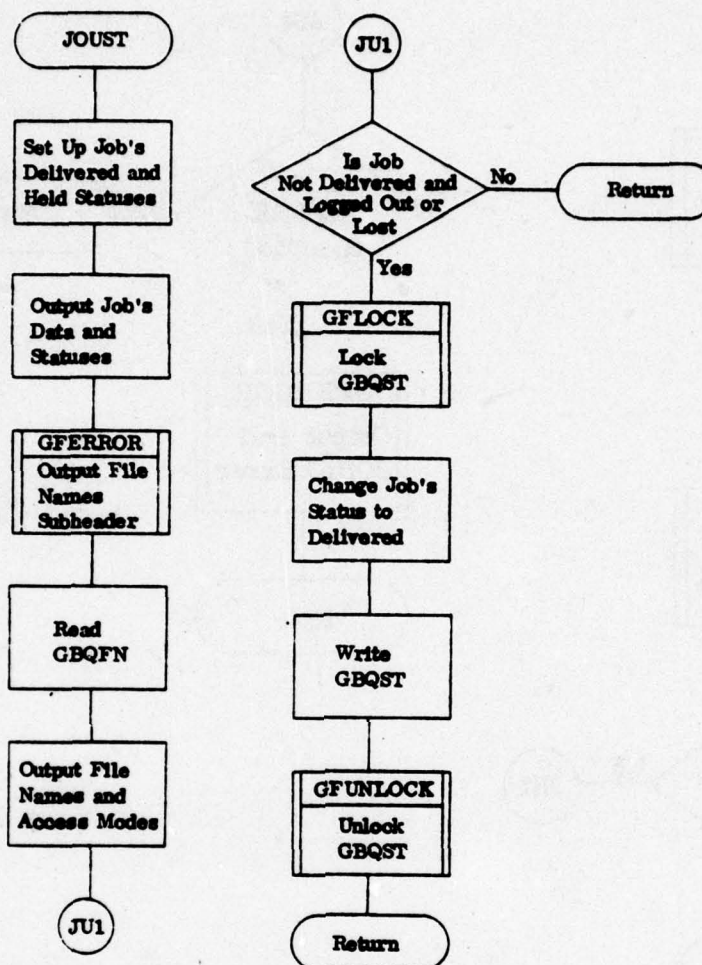


Figure 30. JOBSTAT Process Data Flow (Sheet 3 of 3)

LIST (List a User's Text Files)

The LIST process processes the LIST command. It informs a user of the names and creation dates of his text files.

GLOBAL DATA USAGE - LIST uses the following global data:

GFILE - user text file descriptions.

LOCAL DATA USAGE - LIST uses no pertinent local data.

GENERAL PROCESS FLOW - LIST first opens and reads the first item of GFILE. If the user has no files, a message is output and LIST terminates. Otherwise, header information is output and the number of user files is saved.

Next, for each occupied item of GFILE, the file name and creation date are output.

When all GFILE items have been processed, LIST outputs an end message and terminates. (See figure 31 for data flow.)

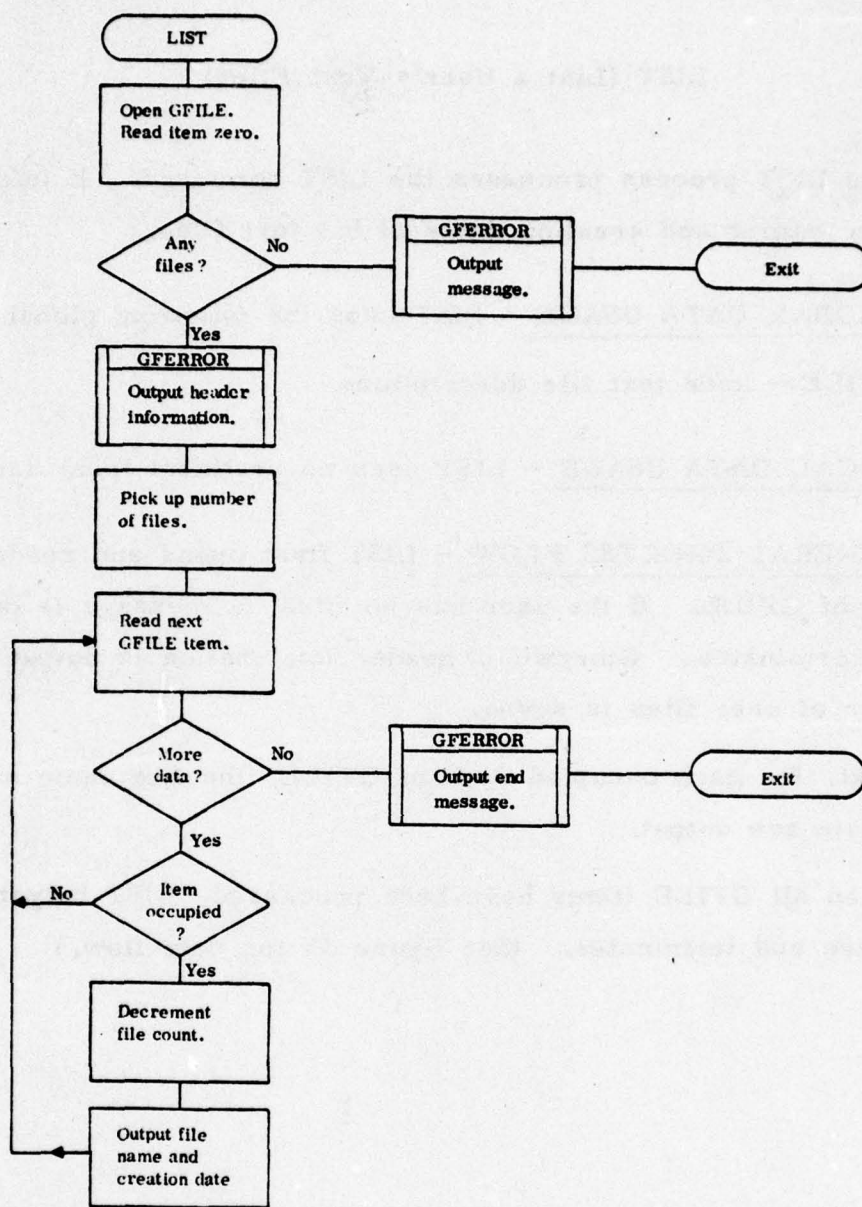


Figure 31. LIST Process Data Flow

LOGOFF (TAS Logoff Process)

The LOGOFF process processes the LOGOFF command. It purges query/responses which have been delivered but not held by the user, and logs the user and his terminal off of TAS.

GLOBAL DATA USAGE - LOGOFF uses the following global data:

- GBQFN - user batch query filenames
- GBQST - user batch query statuses
- GTERM - terminal descriptions
- GUDES - user descriptions

LOCAL DATA USAGE - LOGOFF uses no pertinent local data.

GENERAL PROCESS FLOW - LOGOFF locks (via GFLOCK) and reads the user's GBQST file, and searches for any queries where the response has been 'delivered' and has not been placed in HOLD status by the user. For each such query, if any, LOGOFF unlinks the query and response files and deletes the JOBID entry in the GBQST file and the associated filenames from GBQFN (via the LODELFS function). The user is informed of the deleted files. If GBQST and GBQFN have been modified, LOGOFF writes out both files. LOGOFF then unlocks GBQST via GFUNLOCK.

LOGOFF locks GUDES via GFLOCK and reads in the user's GUDES entry. The user's logon status is set to logged off and the GUDES entry is written out. GUDES is unlocked via GFUNLOCK.

The terminal which the TAS user is using is next logged off TAS in a similar manner. The file GTERM is locked via GFLOCK and read. The terminal's ID is retrieved and its status is set to logged off. The terminal's GTERM entry is written out and the GTERM file is unlocked via GFUNLOCK.

The TAS Logging Process (TLP) is called via a fork/execute sequence to record the logging off event for this user. A Logoff message is output to the user. Once an interrupt signal (DEL) is received from a user, LOGOFF issues a 'KILL' signal to remove any other processes belonging to this user and terminates. (See figure 32 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

LODELFS (Delete the Job's Files) - LODELFS deletes the filenames associated with the batch query job as they are stored in GBQFN. The GBQST.GFPTR entry points to the starting location in GBQFN which is concerned with this job's files. LODELFS positions the GBQFN file accordingly and the file counter is read. For each file stored in GBQFN for this particular batch query job, an empty item is written into GBQFN. Having deleted the filenames from GBQFN, the GBQFN counter is adjusted only if the cleared items were situated at the end of the GBQFN file. If so, the first item of GBQFN is written out.

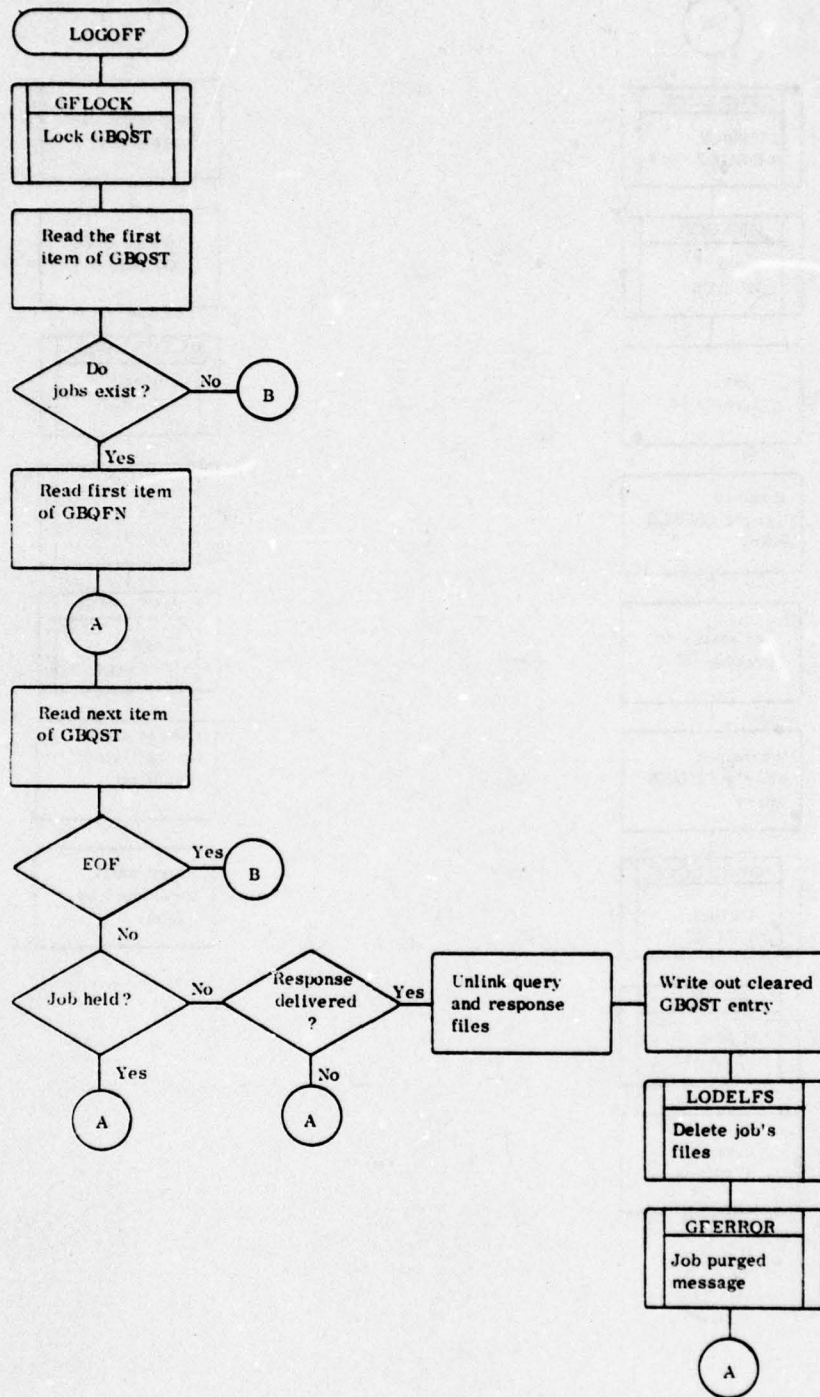


Figure 32. LOGOFF Process Data Flow (Sheet 1 of 3)

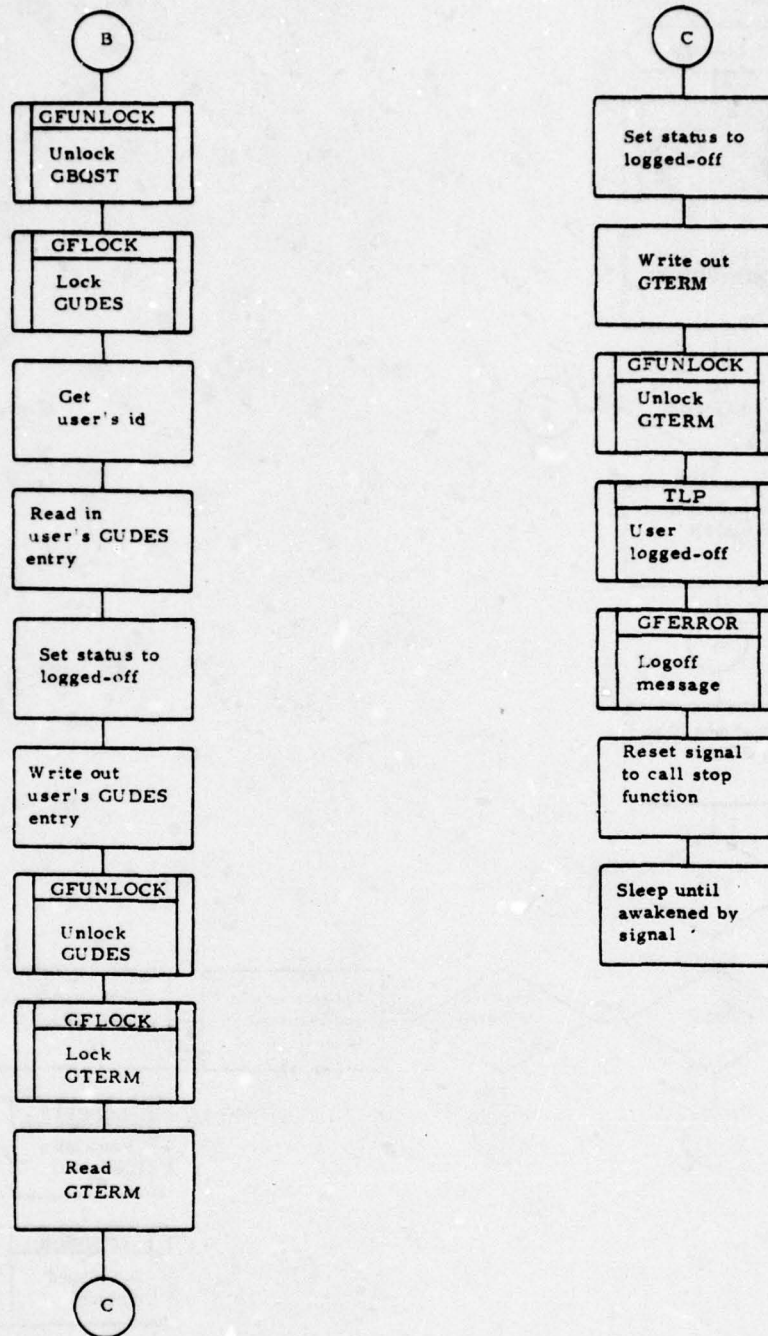


Figure 32. LOGOFF Process Data Flow (Sheet 2 of 3)

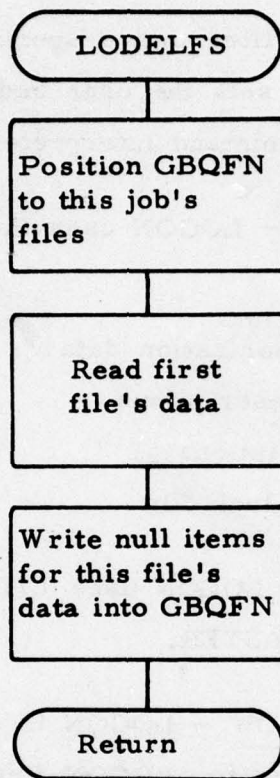


Figure 32. LOGOFF Process Data Flow (Sheet 3 of 3)

LOGON (TAS Logon Process)

The LOGON process verifies that a specified user and terminal are valid under TAS. LOGON sets the user and terminal statuses to logged on and calls the TAS command interpreter (TAS Shell).

GLOBAL DATA USAGE - LOGON uses the following global data:

- GAUTH - user's authorization data
- GTERM - terminal descriptions
- GUDES - user descriptions
- GDOWN - TASDOWN lock file

LOCAL DATA USAGE - LOGON uses LITTY - list of authorized terminals for ROOT and TASTMASTER.

GENERAL PROCESS FLOW - LOGON is called by the UNIX GETTY process with a name input. LOGON first checks to see if the TASDOWN lock file (GDOWN) exists. If it does, then TAS is going down, so LOGON calls LIERROR to output a message to that effect and exit. If the input name starts with a blank character, then LOGON outputs question marks, asks for a name and reads the user name. The user name which LOGON has is then checked against the permissible login names in the UNIX password file '/etc/passwd'. If the user name is invalid, then LIERROR is called to output an invalid sni message, the TAS Logging Process (TLP) is called via a fork/execute sequence with an unsuccessful logon attempt, and LIERROR is called to exit. If the password file indicates that the user has a password,

then LOGON asks for the user's password. The terminal's mode is set to 'no echo' for reading the password so that the password will not be displayed. After reading, the password, LOGON resets the terminal's mode to echo. If the specified password is invalid, then LIERROR is called to output an invalid password message, TLP is called via a fork/execute sequence with an unsuccessful logon attempt, and LIERROR is called to exit.

After the input name and password (if applicable) are validated, then LOGON saves the login directory pathname to pass to the TAS Shell. If the input name is the word 'help', then LOGON determines the program to be called. In this particular case, the name of the program stored in the password file for the name 'help' is the pathname of the HELP program. LOGON performs an execute to the HELP program.

If the user trying to logon is either the ROOT (super-user) or the TASMMASTER, i.e. has user ID zero or one, then the input tty is searched for in LITTY. If the tty is not found, then LIERROR is called to output an 'Invalid TTY for TASMMASTER/superuser' error and exit.

The terminal's ownership is changed (chown) to that of the user logging on and LOGON changes to the login directory (chdir) as specified in the password file. The UNIX accounting files "/etc/utmp" and "/usr/adm/wtmp" are updated if they exist. LOGON then sets the user ID and group ID (setuid, setgid) to those of the user. If the user is the ROOT, then LOGON determines the name of the program to be called. In this particular case, the name of the program stored

in the password file for the superuser is the pathname of the UNIX shell, `"/bin/sh"`. This is done because the default program called by LOGON is the TAS Shell and the superuser expects to have the capabilities of the UNIX shell. LOGON performs an execute to the shell program.

LOGON locks the User Descriptions (GUDES) file via GFLOCK. The GUDES file is then opened and the user's entry is read. If there is no entry for the user or if the GUDES name is not the same as the input name, or if the status of the user is already set to logged on, then LIERROR is called to output an error, TLP is called via a fork/execute sequence with an unsuccessful logon attempt, and LIERROR is called to exit. If the user's authorization data are being modified by his administrative user or the TASMASTER, then LIERROR is called to output a 'Being Updated' message and exit.

LOGON locks the Terminal Descriptions (GTERM) file via GFLOCK. The GTERM file is then opened and read. The input terminal name (UNIX single character designator) is searched for in GTERM. If the terminal is not found, or if the terminal's authorization data are being modified by the TASMASTER, then LIERROR is called to output an error/message and exit. Otherwise, the terminal's status is set to logged on. The terminal's single character name and GTERM index are stored in the user's GUDES entry. GTERM is written, closed and then unlocked via GFUNLOCK. The user's status in GUDES is set to logged on and the project name is set to 'default'. GUDES is then written and closed. LOGON opens and reads the user's Authorization Data (GAUTH) file. If the user's project ID in GAUTH

is not already set to default (zero), then LOGON sets the project ID to default and writes GAUTH back to disk. GAUTH is closed and then GUDS is unlocked via GFUNLOCK. LOGON calls TLP with a successful logon via a fork/execute sequence. LOGON then determines which program is to be called. For TAS users, no program should be specified in the password file and LOGON executes the TAS Shell program. If a different program were specified, LOGON would execute that program. (See figure 33 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

LIERROR (Output Message and/or Exit) – LIERROR is called with an error/message number (or zero) and an exit flag set to either zero or non-zero. If the error/message flag is set, then the specified message is output via GFERROR. If the exit flag is not set, then LIERROR returns. Otherwise, LIERROR unlocks any locked files via GFUNLOCK and then exits.

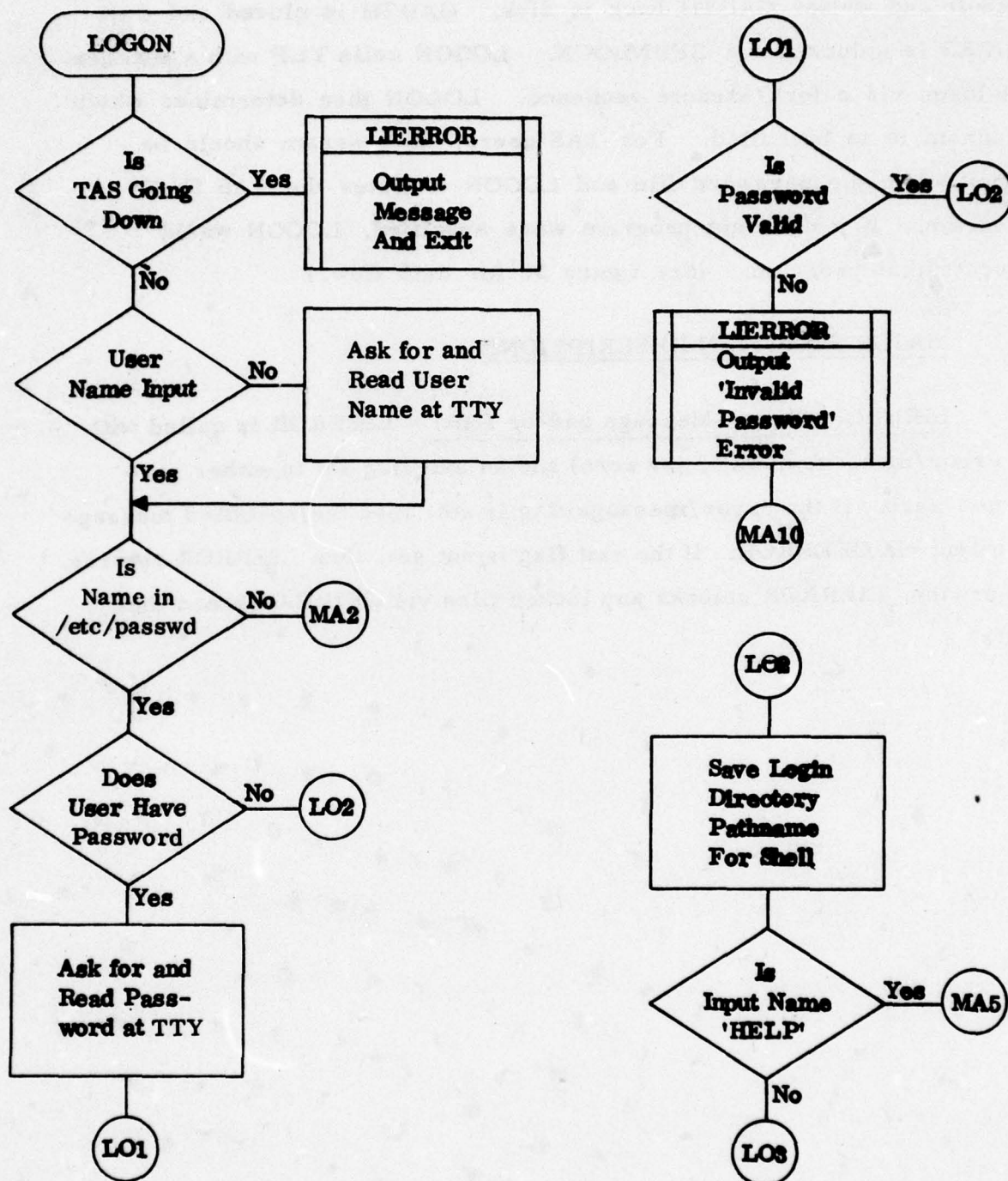


Figure 33. LOGON Process Data Flow (Sheet 1 of 5)

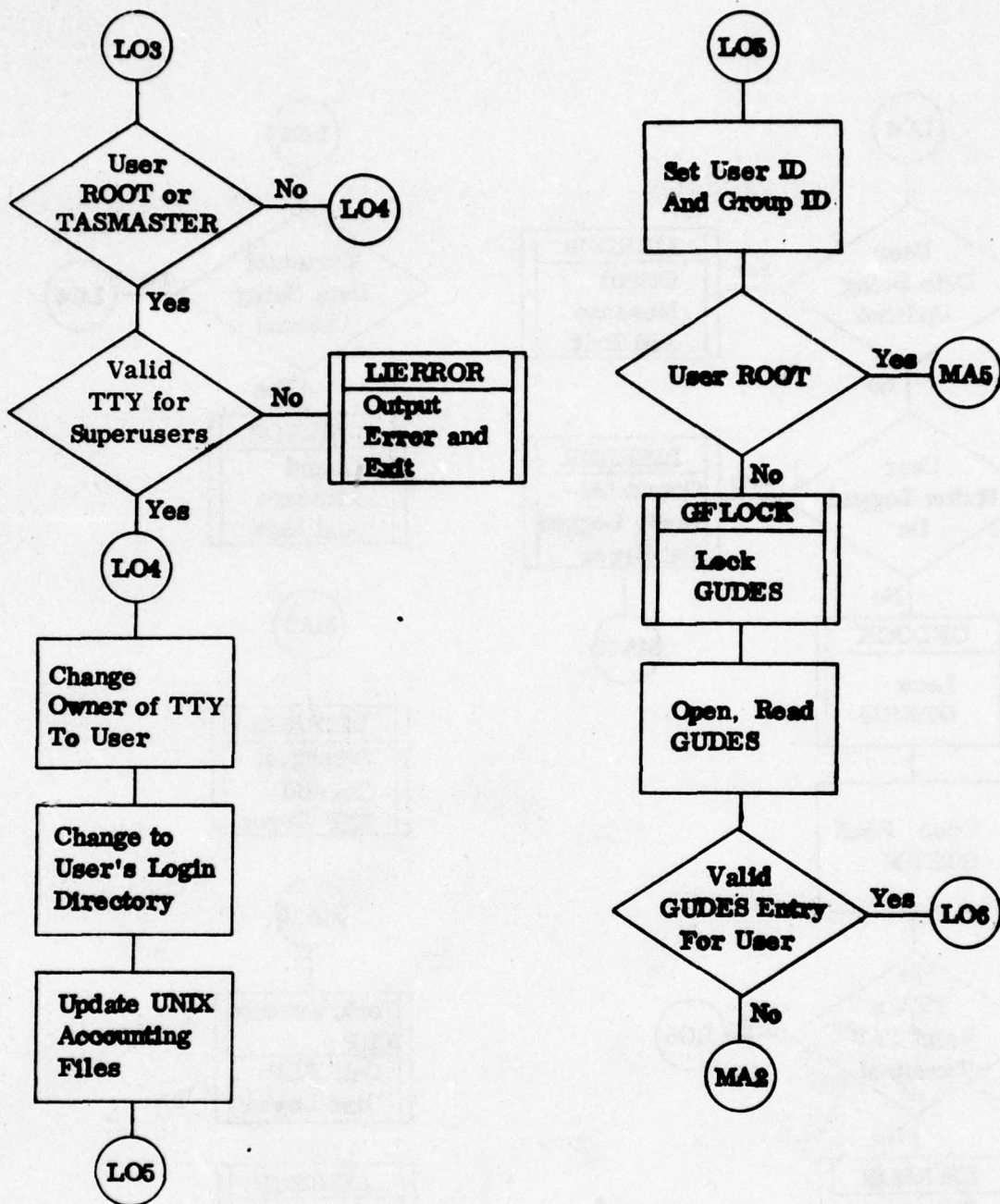


Figure 33. LOGON Process Data Flow (Sheet 2 of 5)

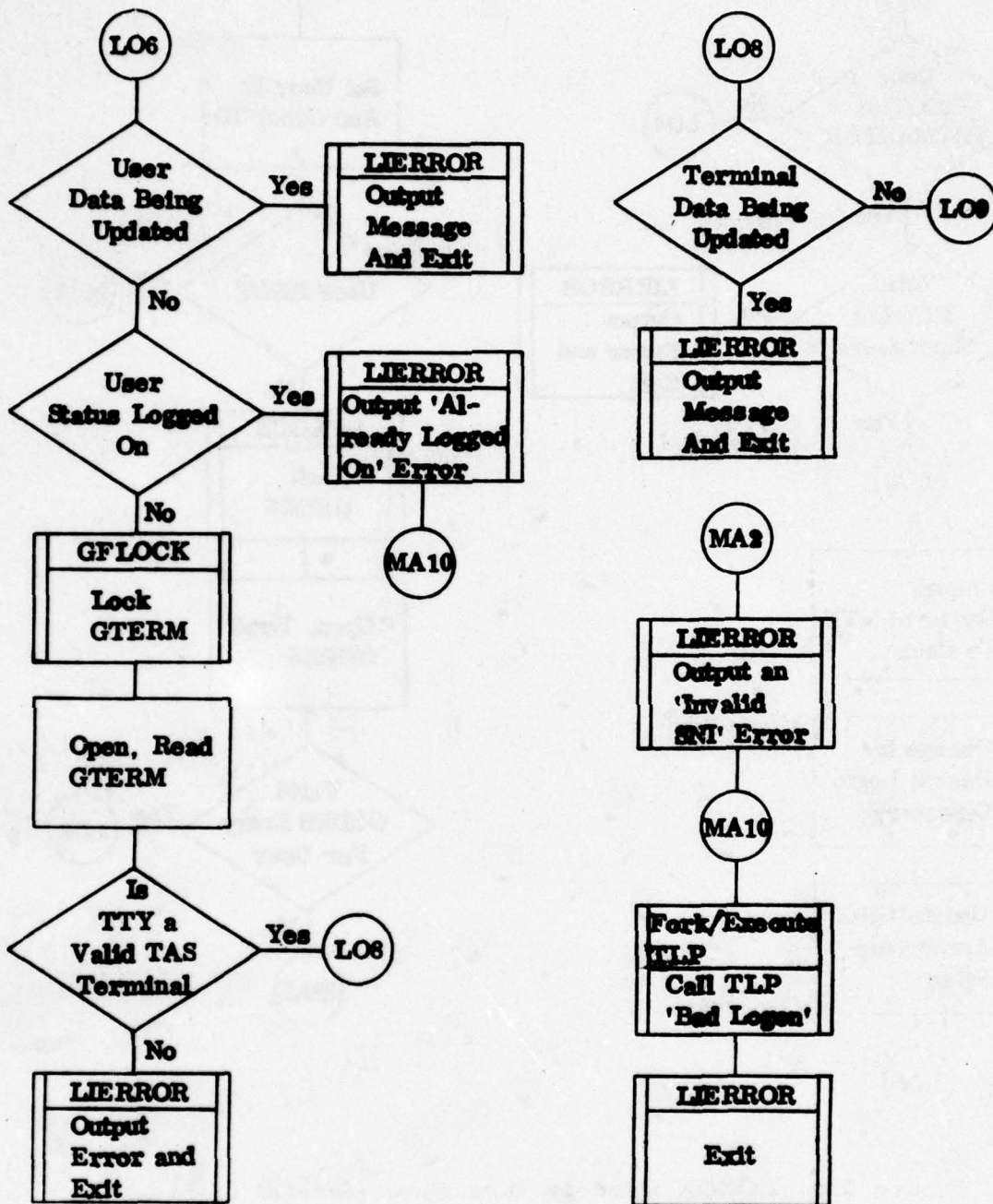


Figure 33. LOGON Process Data Flow (Sheet 3 of 5)

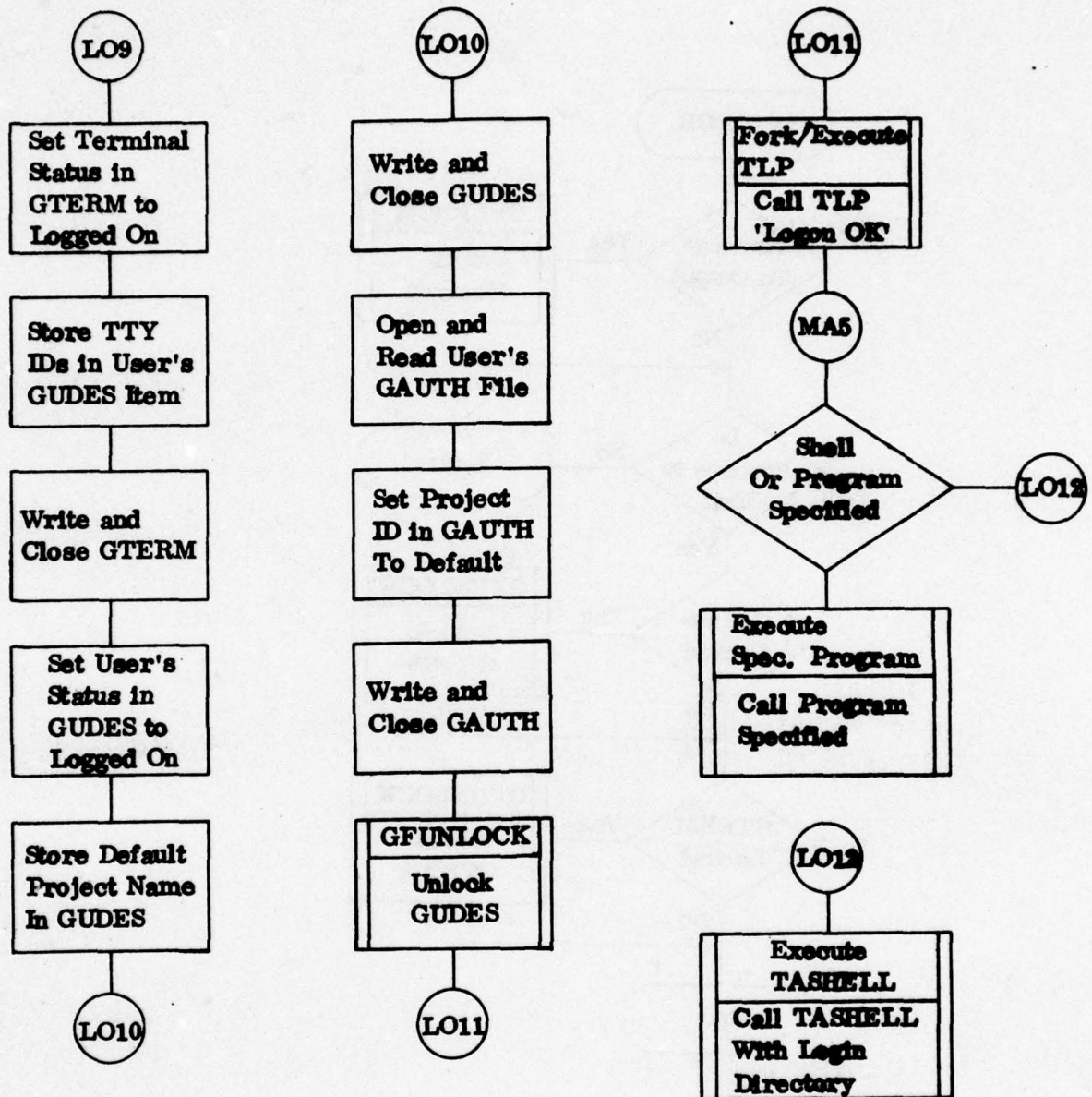


Figure 33. LOGON Process Data Flow (Sheet 4 of 5)

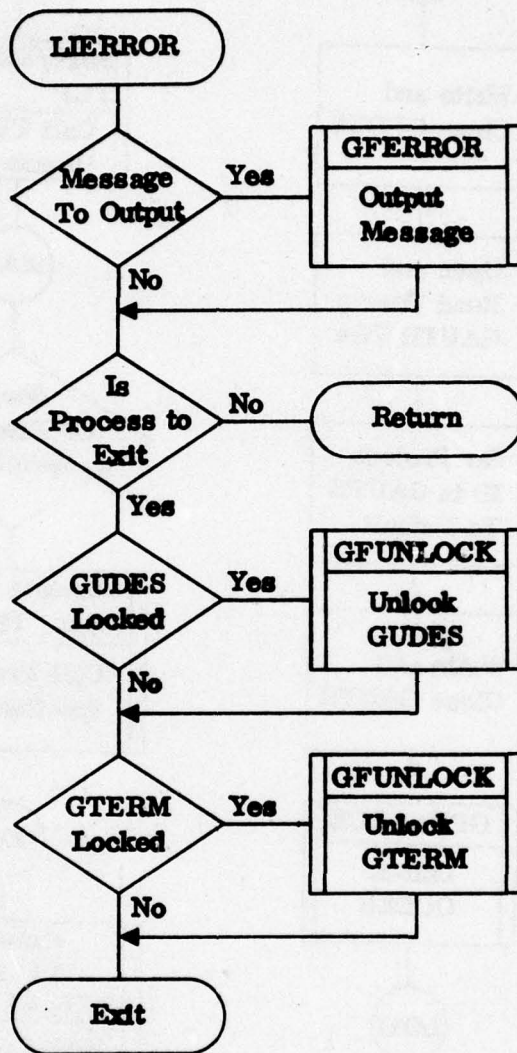


Figure 33. LOGON Process Data Flow (Sheet 5 of 5)

LOGTAPE (Mount a New Logging Tape or Change Logging Tape Mode)

LOGTAPE is the command process which allows the TASMASTER to communicate with the Logging Tape Daemon (LTD) to mount new logging tapes or to change the mode of the logging tape.

GLOBAL DATA USAGE - LOGTAPE uses the following global data:

GLBUF - COINI record buffer

LTPID - logging tape daemon process ID

LOCAL DATA USAGE - LOGTAPE uses no pertinent local data.

GENERAL PROCESS FLOW - LOGTAPE first determines if the user is the TASMASTER. If not, an error message is output and LOGTAPE exits. Then a check is made to ensure that the command was issued from the operator's console (tty8). If not, an error message is output and LOGTAPE exits.

LOGTAPE then sets its real and effective user ID to that of TAS (TUID) and locks GLBUF. LOGTAPE then opens and reads LTPID to obtain the LTD process ID. An "end-tape" signal is then sent to LTD to close down the current logging tape and determine the desired logging mode from the operator. LOGTAPE then issues an end message and terminates. (See figure 34 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

LTERERROR (General error processor) - When called from LOGTAPE, LTERERROR first checks if an error message number was passed as an argument. If so, GFERROR is called to output the error message. If a normal

exit is specified, LTERROR sleeps until a terminating signal is received; otherwise, GFERROR is called to output an end message. Finally, if an error occurred, GFUNLOCK is called to unlock GLBUF before LTERROR exits.

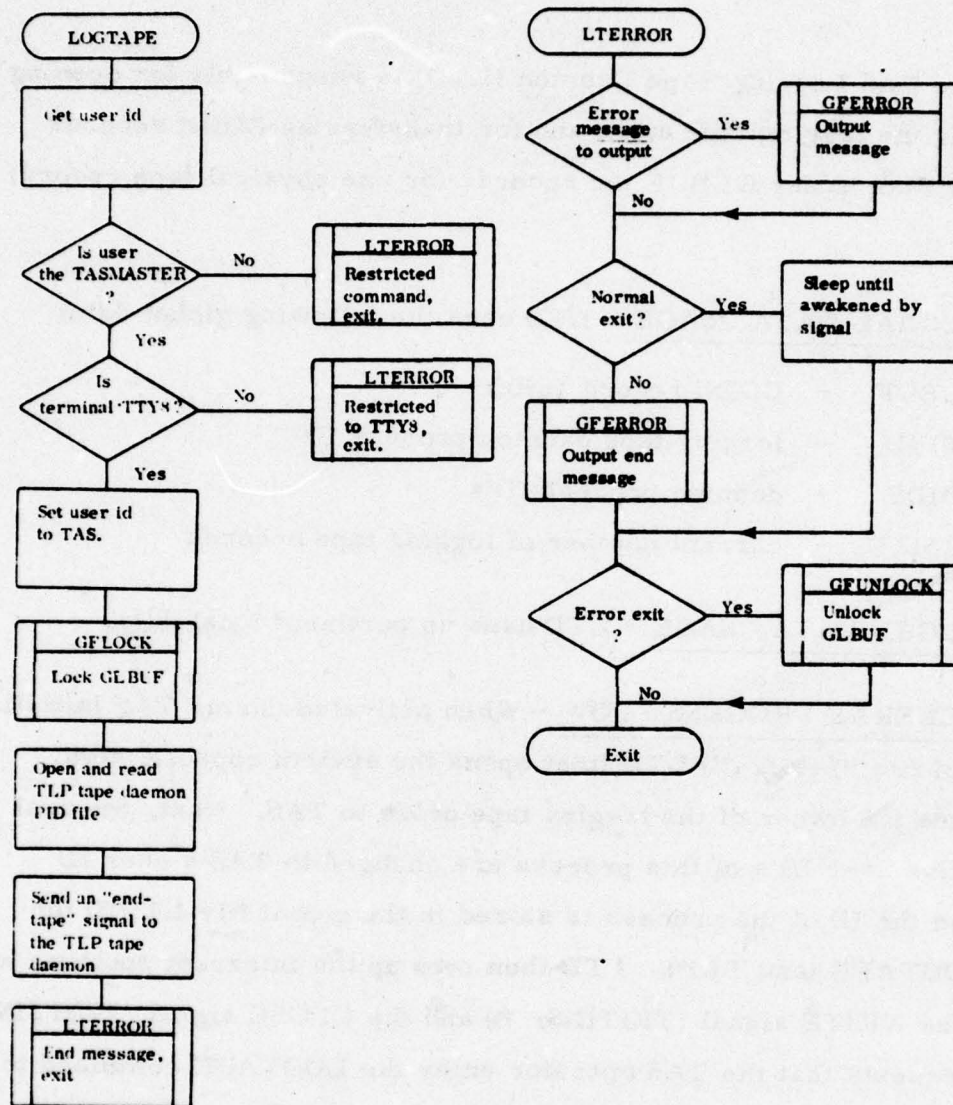


Figure 34. LOGTAPE Process Data Flow

LTD (Logging Tape Daemon)

The TAS Logging Tape Daemon (LTD) is responsible for opening and closing the logging tape drive and for transferring COINI records from their disk buffer GLBUF ten records (or one physical tape record) at a time.

GLOBAL DATA USAGE - LTD uses the following global data:

GLBUF - COINI record buffer
LTPID - logging tape daemon process ID
GPIDS - daemon process ID's
LTSIZE - current number of logging tape records

LOCAL DATA USAGE - LTD uses no pertinent local data.

GENERAL PROCESS FLOW - When activated during TAS initialization (out of file "/etc/rc") LTD first opens the system console, tty8, and changes the owner of the logging tape drive to TAS. Next, the real and effective user ID's of this process are changed to TAS's user ID (TUID) and the ID of the process is stored in the global file LTPID for use by LOGTAPE (and TLP). LTD then sets up the interrupt routines which process the WRITE signal (TIOTINS: 6) and the CLOSE signal (TEMTINS: 7), and requests that the TAS operator enter the LOGTAPE command to mount the logging tape.

LTD then enters an infinite loop of mounting, writing, and unmounting logging tapes. At the start of this loop the mode of operation is set to WAIT and GLBUF is locked via GFLOCK. LTD then waits until 1) a WRITE

signal is received from TLP, or 2) a CLOSE signal is received from the LOGTAPE command.

If a write signal is received from TLP, LTD first checks to see if a logging tape is mounted. If so, the disk buffer for the COINI records, GLBUF, is opened and read. The ten COINI records read are written onto the tape as one physical record and LT_BUMP is called to increment the number of tape records written stored in the file LTSIZE. GLBUF is then closed and recreated and LTD begins the infinite loop again.

If a close signal is received, LTD first checks to see if a logging tape is already mounted. If so, the tape drive is closed and rewound. The current number of physical tape records on the tape (with 10 COINI records per tape record) is then output. LT_MOUNT is then called to interact with the operator to mount a new tape or change the logging mode so that no records are saved on tape. Finally, LTD outputs a message to the operator to hit the delete key to terminate the command, after which LTD begins its infinite loop again. (See figure 35 for data flow.)

MAJOR FUNCTIONS DESCRIPTIONS

LT_MOUNT (Mount a logging tape or change tape mode) - LT_MOUNT first requests the new logging tape mode from the operator, 0 for no logging tape, 1 for a partially written old tape, or 2 for a new tape. The number of records on the tape is set to zero and if the tape is a partially written one, LT_MOUNT requests the number of current blocks on that tape. If a tape is then to be mounted, LT_MOUNT waits until the operator has done so, then calls LT_BUMP to copy the number of tape records onto the file

LTSIZE. If a tape is mounted, LT_MOUNT opens the tape drive, and if it is an old tape, the already written records are read thus positioning the tape at the next available block.

LT_BUMP (Increment the number of tape records in LTSIZE) -

LT_BUMP first opens the file LTSIZE and copies its input argument, the current number of tape blocks, onto the file LTSIZE in ASCII format, then closes LTSIZE. If the maximum number of records for one tape have been written, the tape drive is closed, a "tape full" message is output, and LT_BUMP requests that the operator enter the LOGTAPE command to mount a new tape.

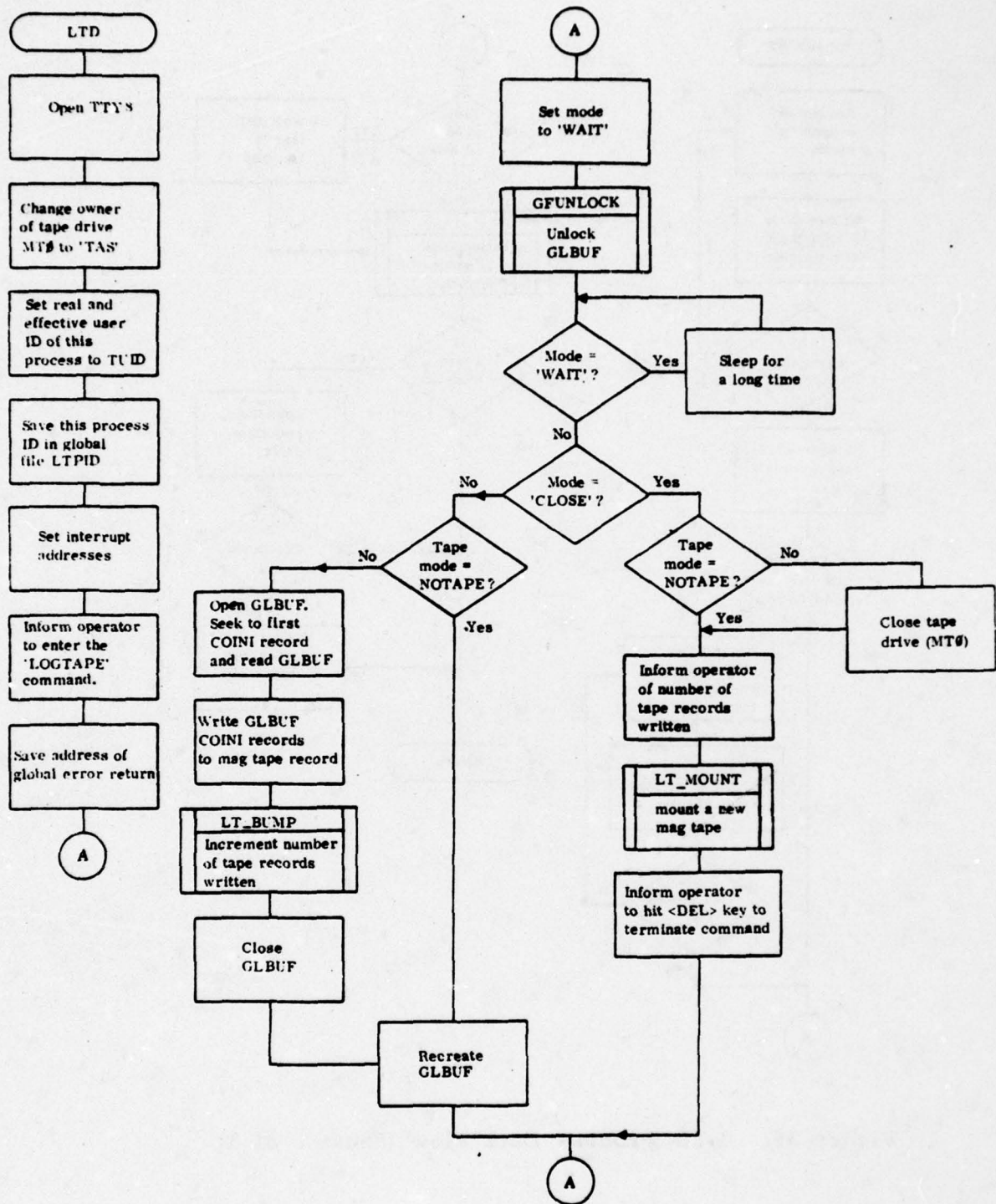


Figure 35. LTD Process Data Flow (Sheet 1 of 3)

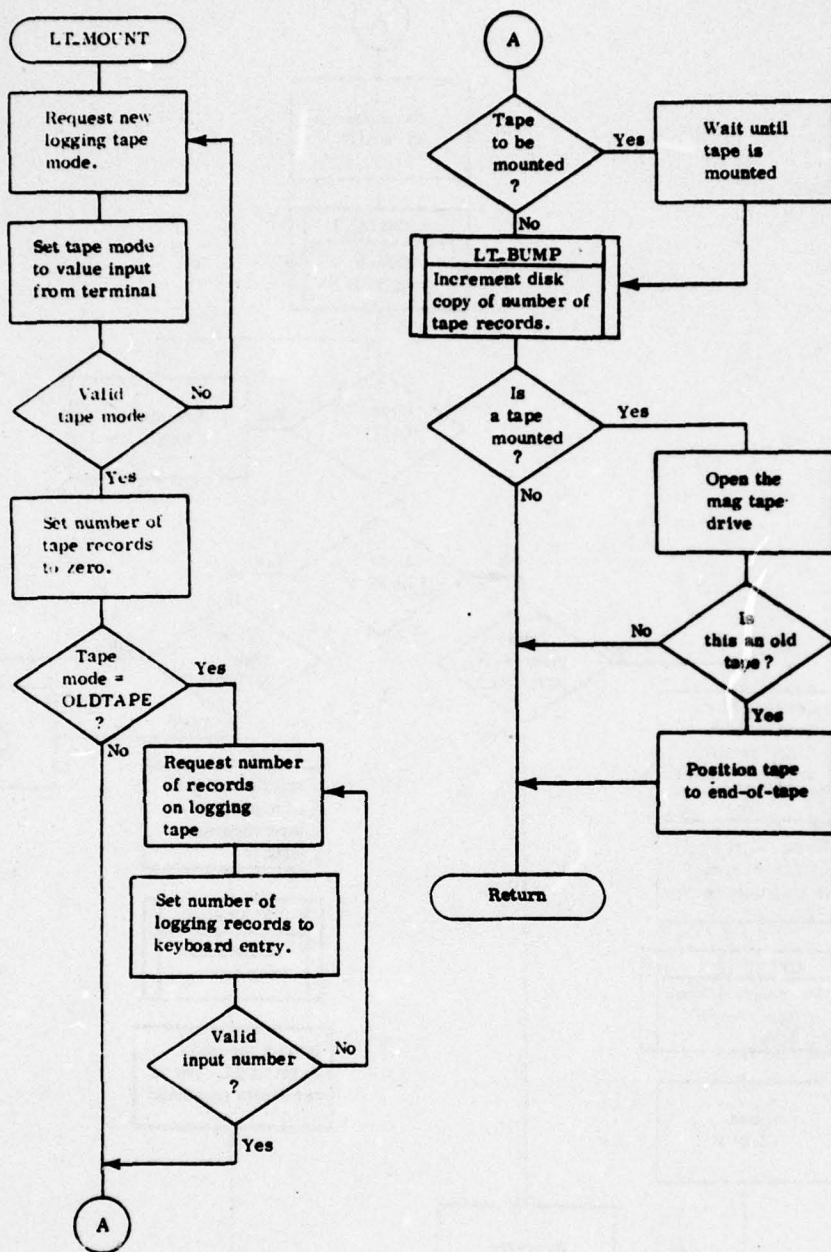


Figure 35. LTD Process Data Flow (Sheet 2 of 3)

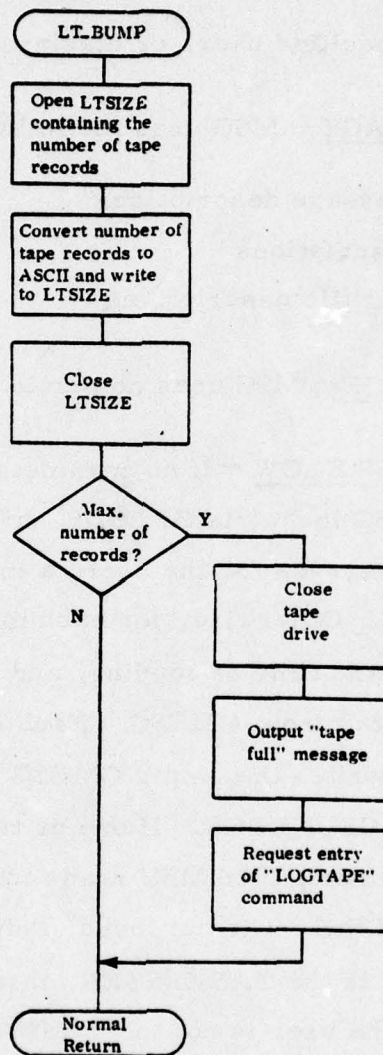


Figure 35. LTD Process Data Flow (Sheet 3 of 3)

MSG (User Message Process)

The MSG process processes the MSG command. It allows a user to send a message to another specified user, or displays a user's messages to him.

GLOBAL DATA USAGE - MSG uses the following global data:

- GMESG - user message descriptions
- GUDES - user descriptions
- GFILE - user text file descriptions

LOCAL DATA USAGE - MSG uses no pertinent local data.

GENERAL PROCESS FLOW - If no parameter has been input with the MSG command, then MSG locks via GFLOCK and reads the user's GMESG file. If there are no messages for the user, a message to that effect is output and MSG terminates. Otherwise, for each message in GMESG, the message's originator, date and time of sending, and text are output to the user and the message is deleted from GMESG. Each displayed message's text file is also unlinked. MSG writes the empty GMESG file out and terminates. GMESG is unlocked via GFUNLOCK. If one or two parameters have been input with the MSG command, then MSG reads the GUDES file and searches for the specified sni. If the sni is not found, a diagnostic is output and MSG terminates. If the user is the TASMASTER, then the sni must be any user's sni except his own. If the user is not the TASMASTER, then the sni must be that of the TASMASTER. If the user/sni is any other combination, then MSG outputs a diagnostic and terminates. If a second input parameter is present, then MSG reads the user's GFILE file and searches for the specified filename. If the filename is not found, a diagnostic is output and MSG

terminates. Otherwise, the user's file is read into core and is ready to be used as a message. If no filename is specified, then the user's terminal input is read until an end-of-text (EOT) character is encountered (CONTROL-D). The GMESG file for the specified user ID is locked via GFLOCK and read, and the new message is added to GMESG. The message data from the terminal or from a user file is written into a message text file for the specified user ID. The GMESG file is written and unlocked via GFUNLOCK, and MSG terminates. (See figure 36 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

MSERROR (Output Error and/or Exit) – MSERROR is called with an error number or zero and an exit flag set to either zero or non-zero. If an error number is specified, then the specified error message is output via GFERROR. If the exit flag is not set, then MSERROR returns. Otherwise, if GMESG is locked, it is unlocked via GFUNLOCK, and MSERROR exits.

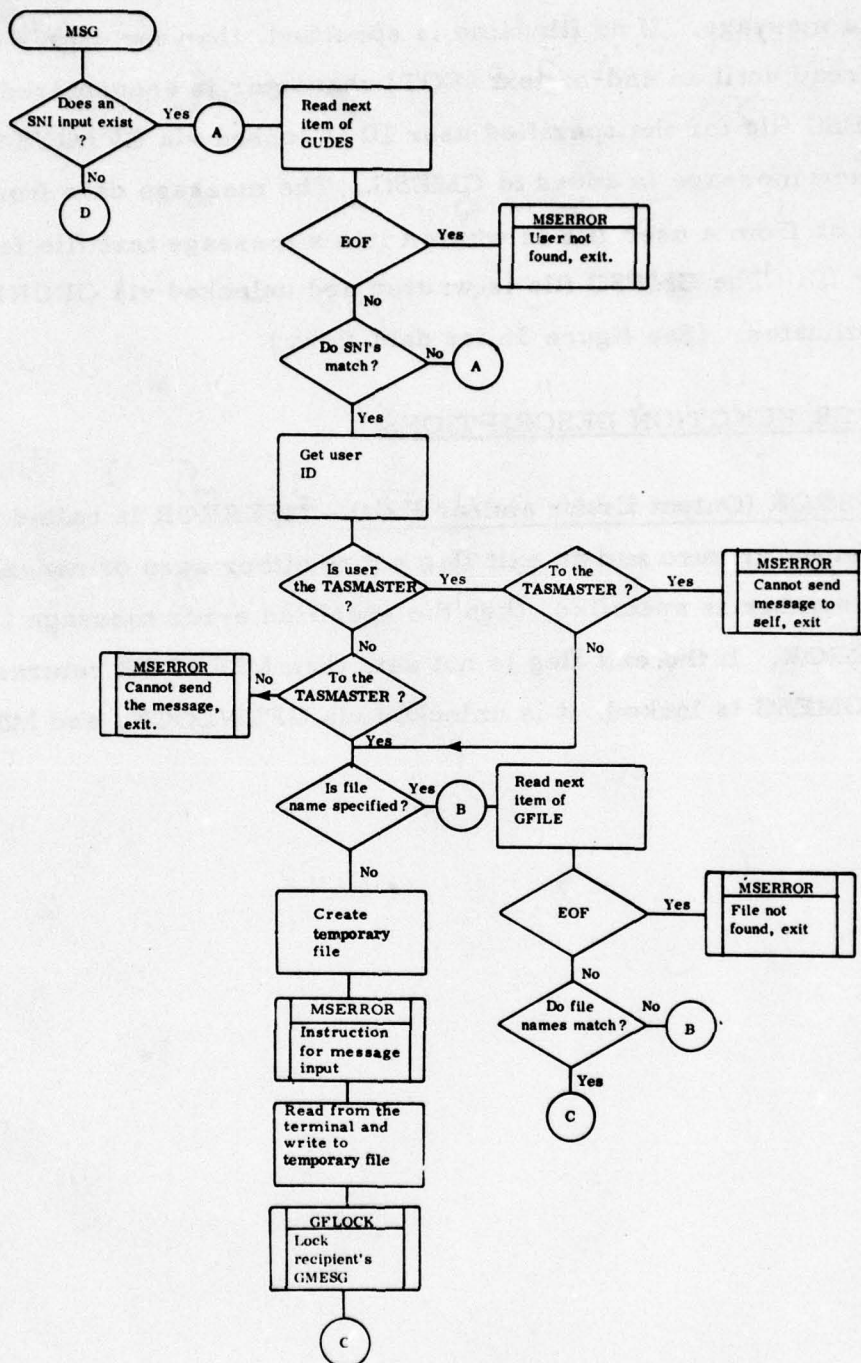


Figure 36. MSG Process Data Flow (Sheet 1 of 3)

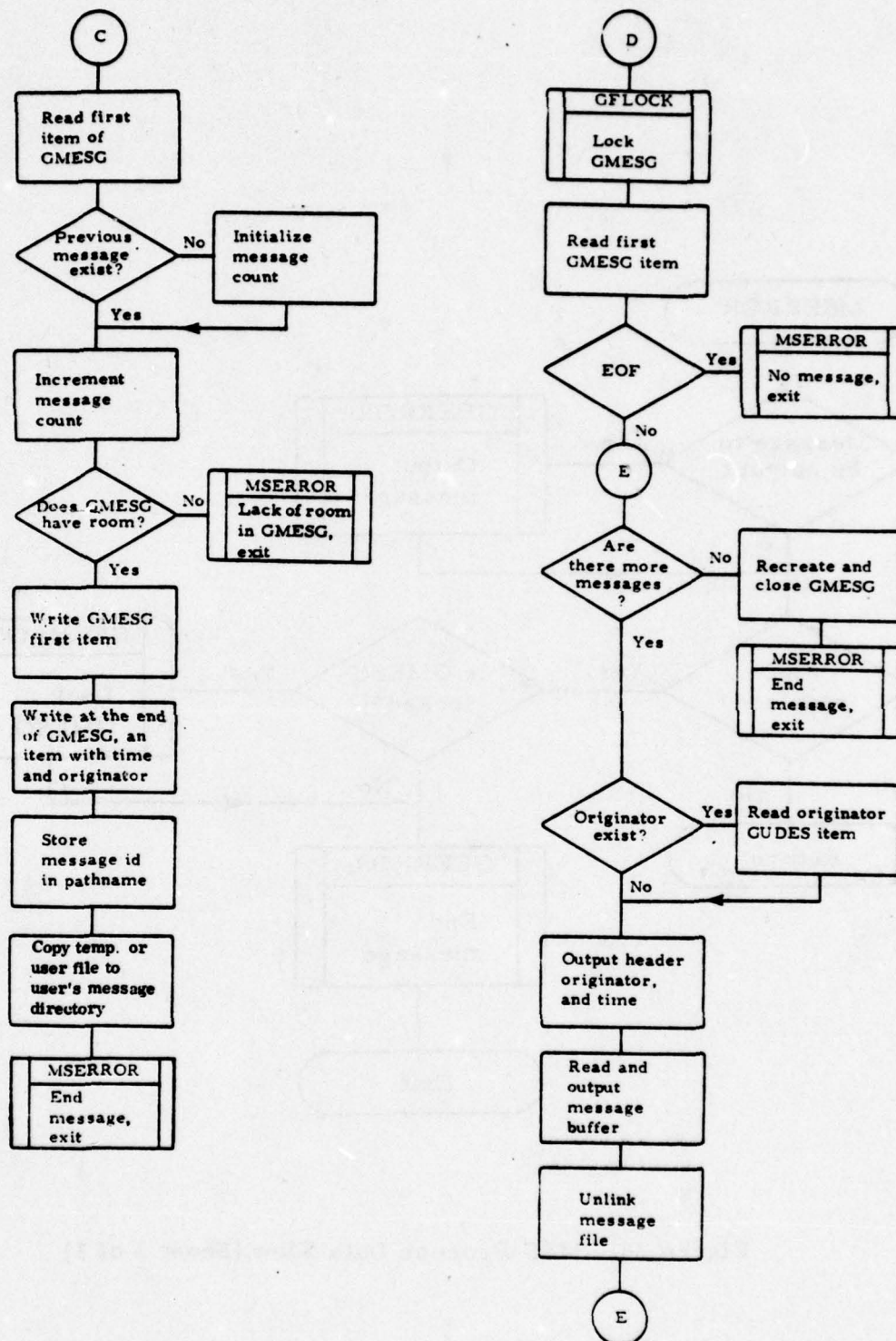


Figure 36. MSG Process Data Flow (Sheet 2 of 3)

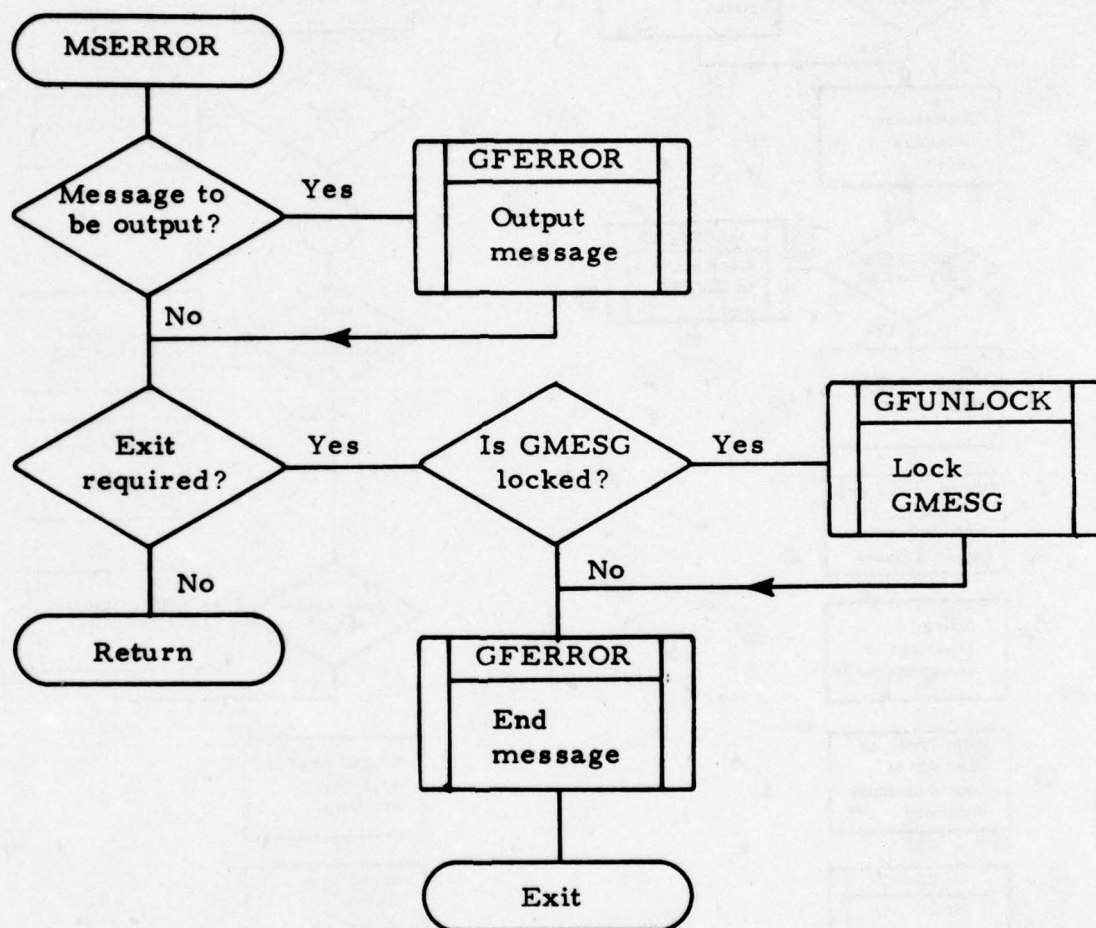


Figure 36. MSG Process Data Flow (Sheet 3 of 3)

PROJECT (Display and/or Set a User's Current Project)

The PROJECT process processes the PROJECT command. It allows a user to determine the name of his current project and/or change his current project.

GLOBAL DATA USAGE - PROJECT uses the following global data:

GAUTH - user authorization data

GPROJ - project descriptions

GUDES - user descriptions

LOCAL DATA USAGE - PROJECT uses no pertinent local data.

GENERAL PROCESS FLOW - PROJECT ensures that the user is not the TASMASTER. If he is, an error message is output and PROJECT terminates. Otherwise, the user's GAUTH file is opened and read, and if no project name was specified as an input argument, the project ID is set to zero for the default project.

If a project name was specified as an input argument, and the user has no projects available to him, an error message is output and PROJECT terminates. Otherwise, the project name is saved, and GPROJ is opened. GPROJ is then searched for the specified project name. If it is not found, an error message is output and PROJECT terminates. Otherwise, the project ID of the matching project is saved.

When the project ID has been established, a test is made to determine if the user is currently under that project. If he is not, PROJECT ignores interrupt signals, locks GUDES, stores the new project name in the user's GUDES item, and stores the new project ID in the user's GAUTH file.

Finally, the project name is saved and output to the user before
PROJECT terminates. (See figure 37 for data flow.)

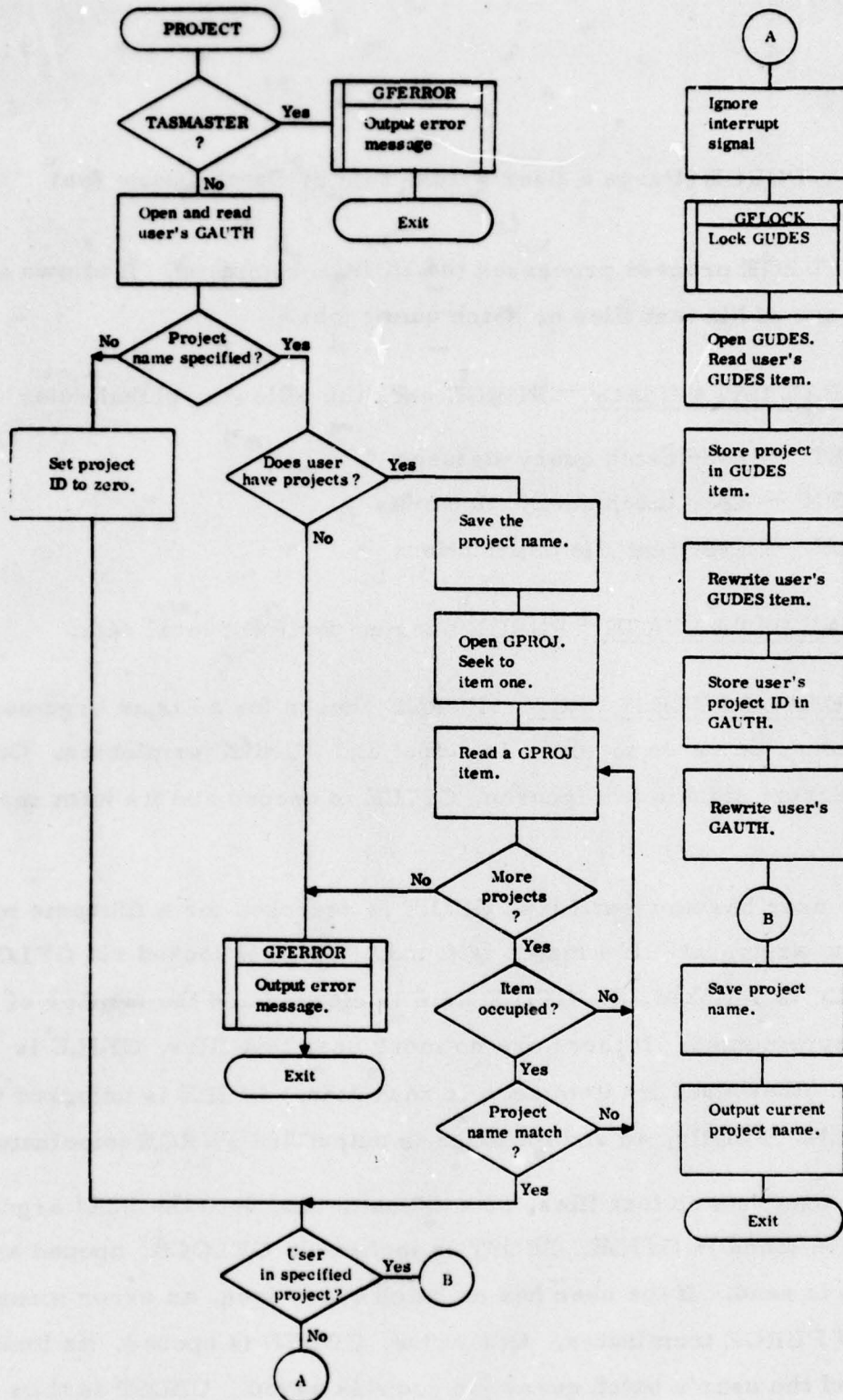


Figure 37. PROJECT Process Data Flow

PURGE (Purge a User's Text File or Batch Query Job)

The PURGE process processes the PURGE command. It allows a user to delete one of his text files or batch query jobs.

GLOBAL DATA USAGE - PURGE uses the following global data:

GBQST - user batch query statuses
GBQFN - user batch query filenames
GFILE - user text file descriptions

LOCAL DATA USAGE - PURGE uses no pertinent local data.

GENERAL PROCESS FLOW - PURGE checks for an input argument. If there is none, an error message is output and PURGE terminates. Otherwise, interrupt signals are ignored, GFILE is opened and its item zero is read.

If the user has any text files, GFILE is searched for a filename match to the input argument. If a match is found, GFILE is locked via GFLOCK, the text file is unlinked, the GFILE item is cleared and the number of user files is decremented. If there are no more user text files, GFILE is recreated; otherwise, its item zero is rewritten. GFILE is unlocked via GFUNLOCK. Finally, an end message is output and PURGE terminates.

If the user has no text files, or a filename match to the input argument could not be found in GFILE, GBQST is locked via GFLOCK, opened and its item zero is read. If the user has no batch query jobs, an error message is output and PURGE terminates. Otherwise, GBQFN is opened, its item zero is read and the user's batch query job count is saved. GBQST is then

searched for a JOBID match to the input argument. If it is not found, an error message is output and PURGE terminates.

If a match is found in GBQST to the input argument, and the job has not been delivered, an error message is output and PURGE terminates. Otherwise, the query and response files are unlinked and the GBQST job count is decremented. If there are no more batch query jobs for this user, GBQST and GBQFN are recreated; otherwise, the GBQST item for this job is cleared, and PUDELFS is called to remove the query's filenames from GBQFN. Finally, an end message is output and PURGE terminates. (See figure 38 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

PUDELFS (Delete a batch job's filename items from GBQFN) - PUDELFS first positions GBQFN to the first item for the specified job and reads it (number of additional filename items). This and successive GBQFN items are cleared until all filename items for this job have been removed. If the items have been removed from the end of GBQFN, the pointer to the next available item (in GBQFN item zero) is decremented and item zero is rewritten. PUDELFS then returns.

PUERROR (Output Message and/or Exit) - PUERROR is called with an error/message number (or zero) and an exit flag set to either zero or non-zero. If the error/message flag is set, then the specified message is output via GFERROR. If the exit flag is not set, then PUERROR returns. Otherwise, PUERROR unlocks any locked files via GFUNLOCK and then exits.

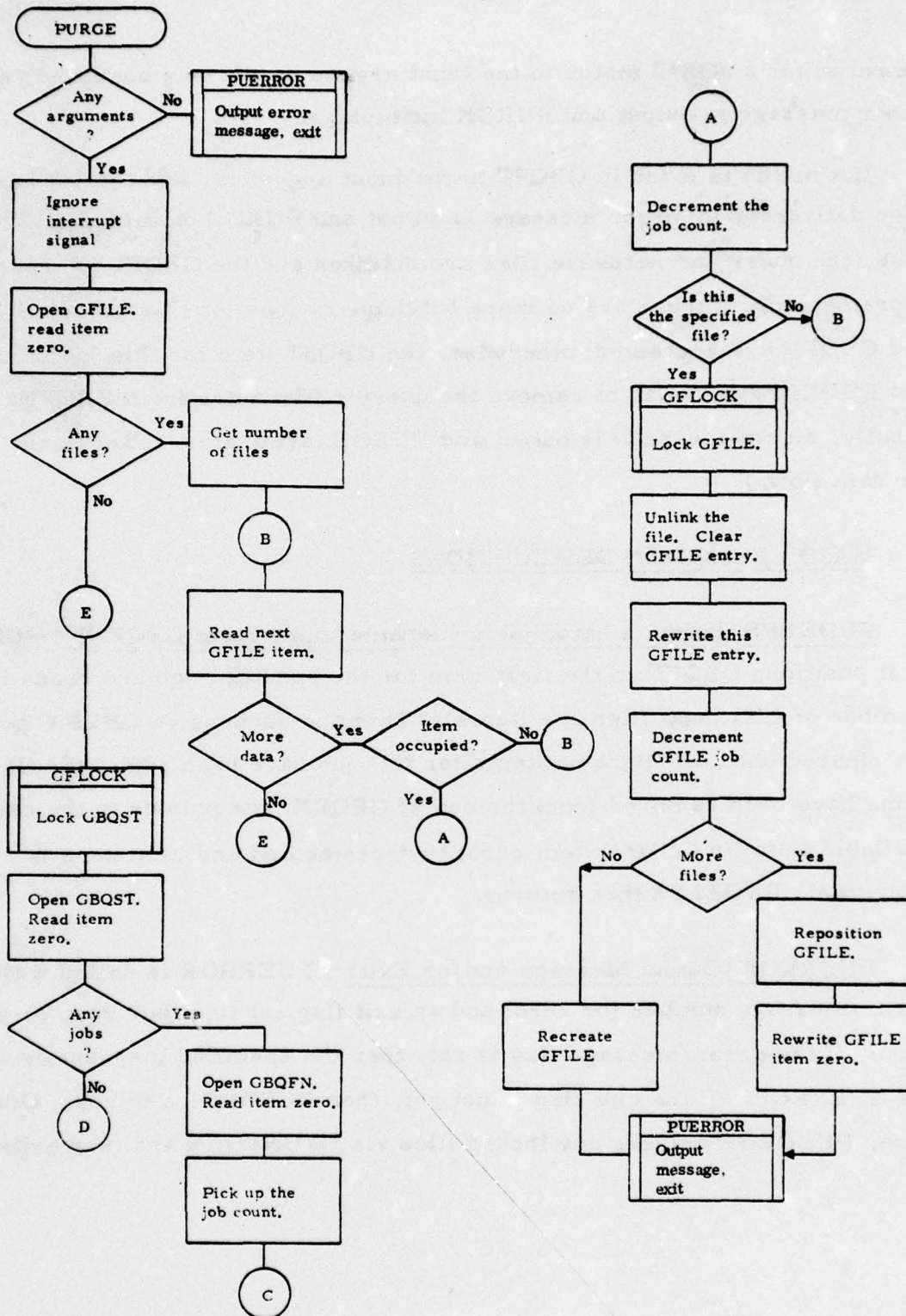


Figure 38. PURGE Process Data Flow (Sheet 1 of 4)

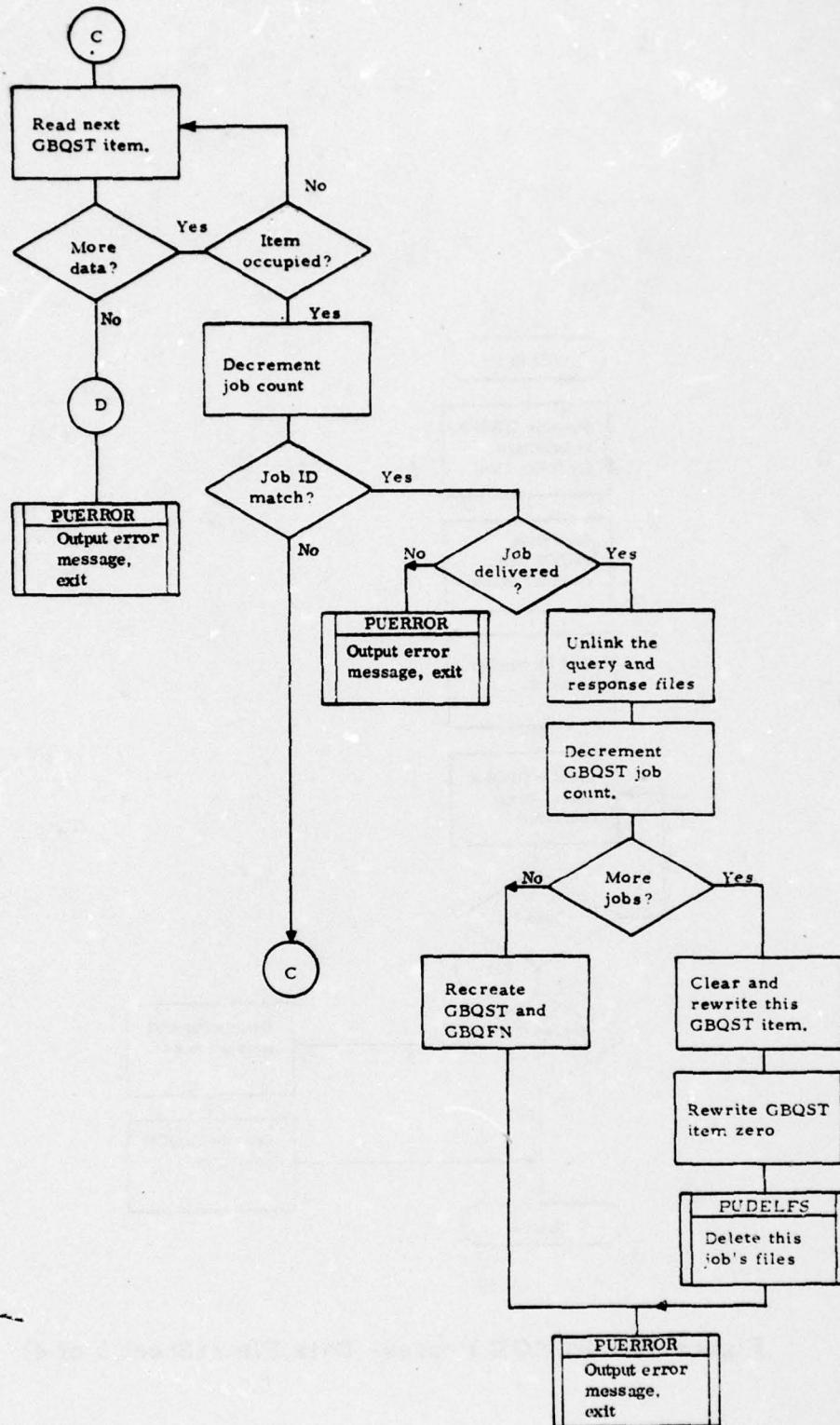


Figure 38. PURGE Process Data Flow (Sheet 2 of 4)

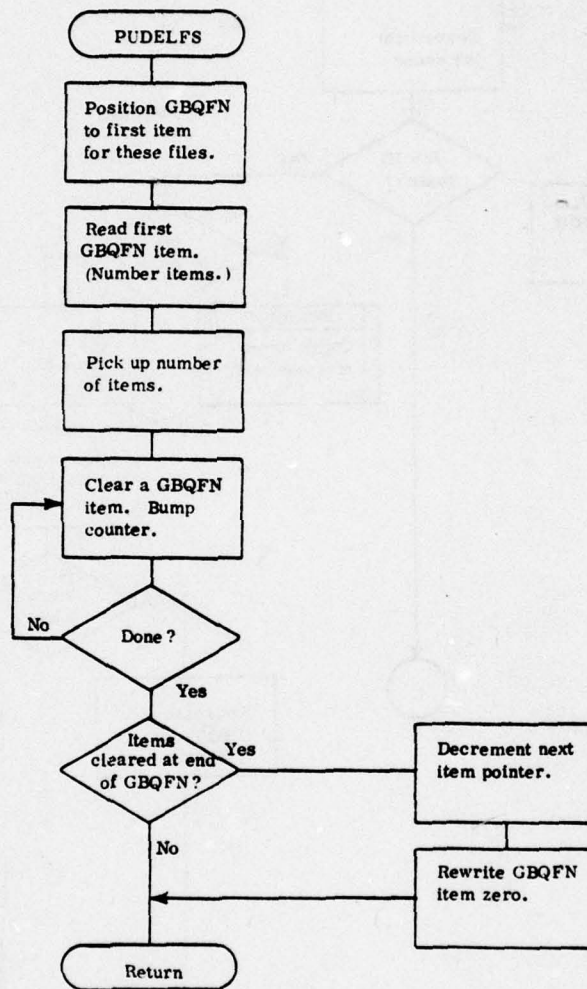


Figure 38. PURGE Process Data Flow (Sheet 3 of 4)

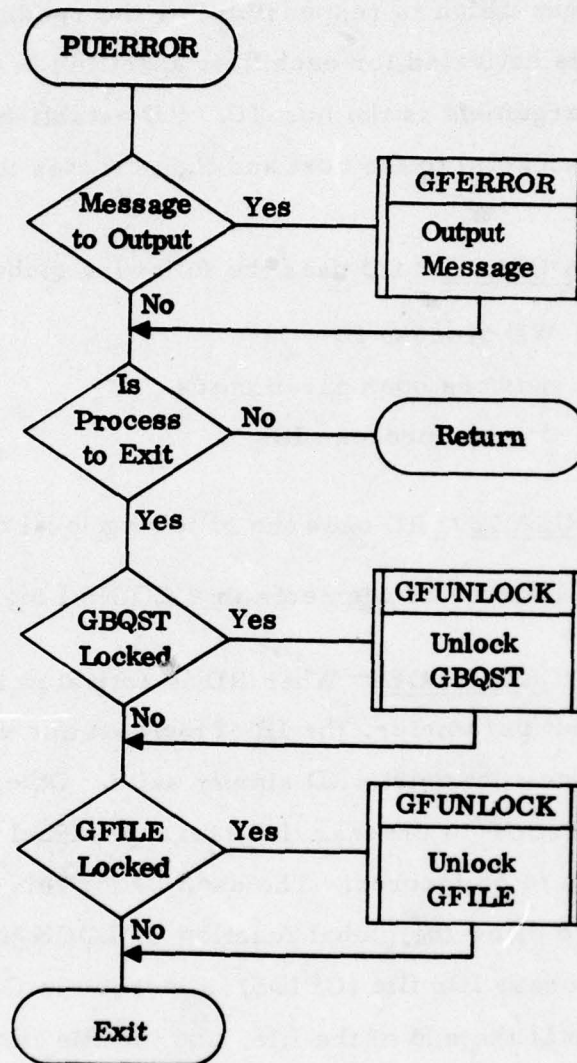


Figure 38. PURGE Process Data Flow (Sheet 4 of 4)

RD (Network Read Daemon)

RD is the process which is responsible for the reading of data from a network host. RD is activated for each host specified in the UNIX "/etc/rc" file. Its input argument is the host ID. RD establishes the read and write connections (sockets) to the host and then creates the INI write daemon (WD).

GLOBAL DATA USAGE - RD uses the following global data:

GIPID - WD process ID
GOPARAMS - network open parameters
GPIDS - daemon process IDs

LOCAL DATA USAGE - RD uses the following local data:

RDWSEGS - Number of segments in a COINS I message

GENERAL PROCESS FLOW - When RD is activated it ensures the existence of one input parameter, the ID of the host for which the daemon is responsible. If it does not exist, RD simply exits. Otherwise, RD establishes function RDHNGUP to process the interrupt signal HANGUP, and that all other signals are to be ignored. The user ID for this process is set to the TAS ID (TUID) to allow the global function GFLOCK to work effectively. Next the daemon process IDs file (GPIDS) is locked via GFLOCK, the process's PID is written at the end of the file, and the file is unlocked via GFUNLOCK.

The read open packet for this host is established in the structure GOPARAMS. Likewise, the write open packet for the host is set up. The opening of the specific host's write socket is attempted every ten seconds

until the open is successful. The procedure is repeated for the host's read socket. Once both sockets are established, the WD process (Network Write Daemon) is activated. The WD process is sent two parameters: the first is the host ID and the second is the host's write socket file descriptor. Having activated the WD daemon, the file descriptor for the host's read socket connection is then duplicated.

The temporary output file for this host is locked and RD reads the next COINS I segment from the read socket. If the read is not successful, the following four-step procedure is followed:

1. The function RDINI calls the INI process with a read error.
2. The host files are closed.
3. The temporary output file is unlocked.
4. The WD is killed via a KILL signal.

The program process flow then returns to the opening of the host's write socket and the RD process again tries to establish communication with the network host.

RD reads COINS I message segments and stores them in the temporary output file. Once the start of a message is read and recognized, the output file is created, the segment is written into it and the segment count, RDWSEGS, is initialized. As each COINS I message is read and it is determined that the segment is not the start of another message, the message segment is appended to the end of the temporary file and the segment count, RDWSEGS, is incremented. The process is repeated until an end-of-message has been received, at which time the temporary file is closed and the function

RDINI is called to fork/execute INI. The function code sent to INI is one of message received. (Note: INI unlocks the temporary output file.) Only those messages which have definitive beginnings and ends are sent to INI to be further processed. Once the function RDINI returns, RD locks the temporary output file and repeats its infinite loop of reading COINS I network messages.

Two special, non-system problems can occur in the collecting and storing of COINS I messages. If a COINS I segment is the start of a message and if another message has already been started, the temporary output file is simply closed and then recreated, and the new message is written out to the output file. If the COINS I message becomes too large, i. e., RDWSEGS greater than 350, the temporary output file is closed and the message is essentially dropped. RD then continues to read from the network until the start of another message is read. At that point the temporary output file is recreated and the new message is processed as described earlier. (See figure 39 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

RDHNGUP (Process Hangup (Exit) Signal) - When RD receives a hangup signal, RDHNGUP resets the HANGUP signal, the temporary output file is unlocked and the RD process exits.

RDINI (Call INI) - The RDINI function calls the INI process with an input function and a host ID. RDINI first performs a UNIX fork call. The parent process then waits for the child to terminate. The child process performs a second fork call. The parent of this fork exits, thus reactivating the original waiting parent. The child of the second fork performs an execute call to INI

passing it the two parameters specified above. The original parent, after being reactivated, returns to the calling function. The reason for the double fork sequence is to allow RD to continue processing without the need to wait for INI to finish in order to clear the INI process ID out of the UNIX tables. The second fork with no wait causes the INI parent process ID to be 'lost' and set to one (the process ID of INIT, which is always waiting).

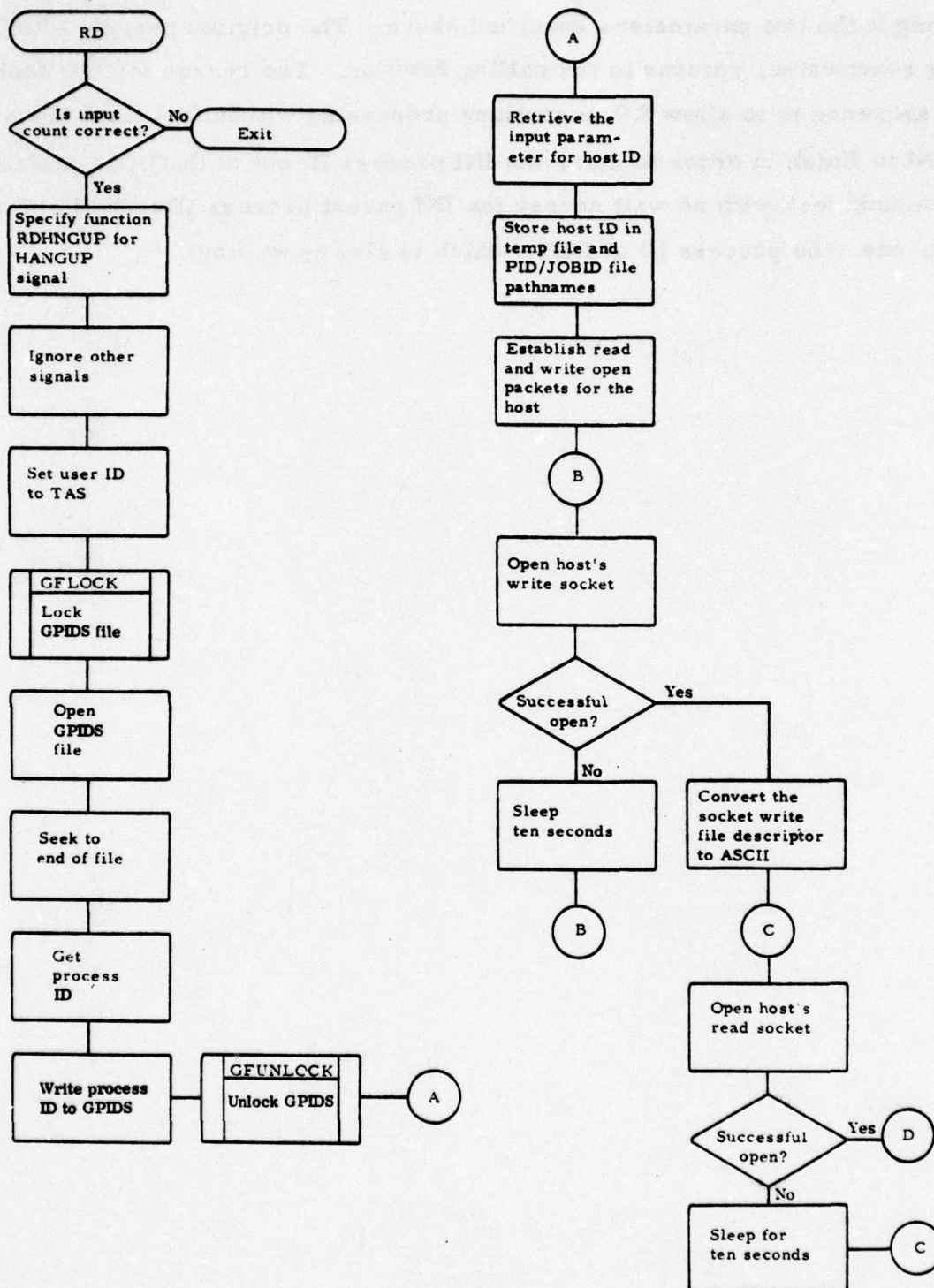


Figure 39. RD Process Data Flow (Sheet 1 of 4)

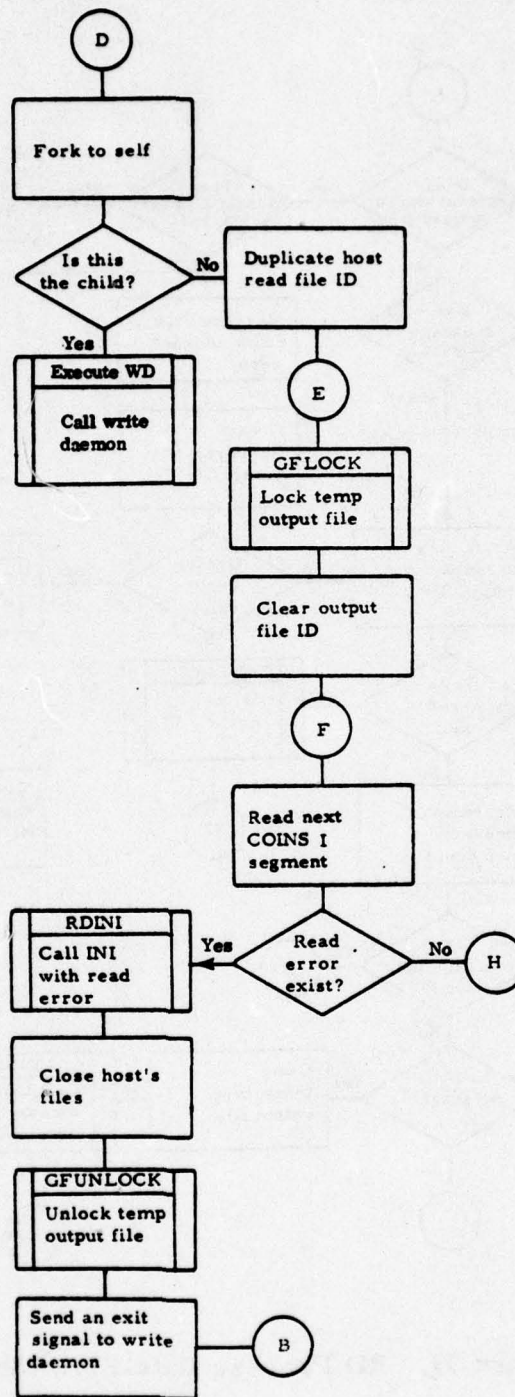


Figure 39. RD Process Data Flow (Sheet 2 of 4)

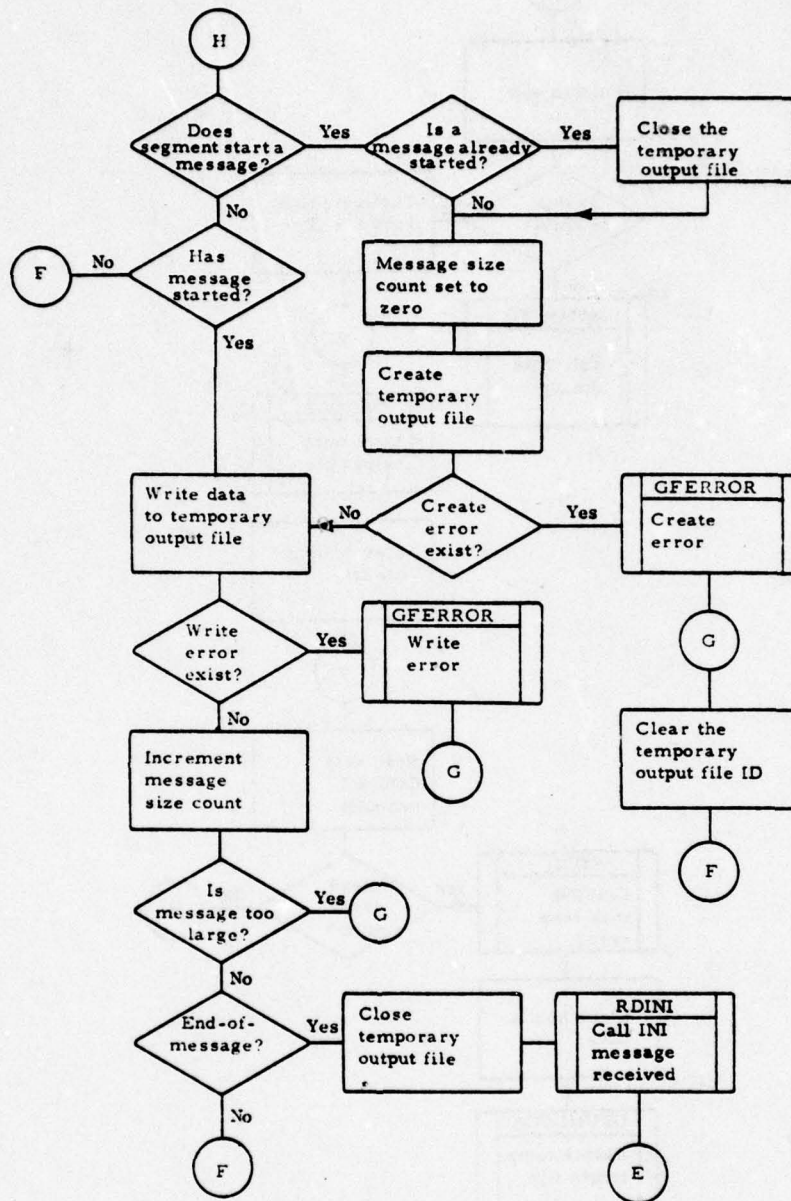


Figure 39. RD Process Data Flow (Sheet 3 of 4)

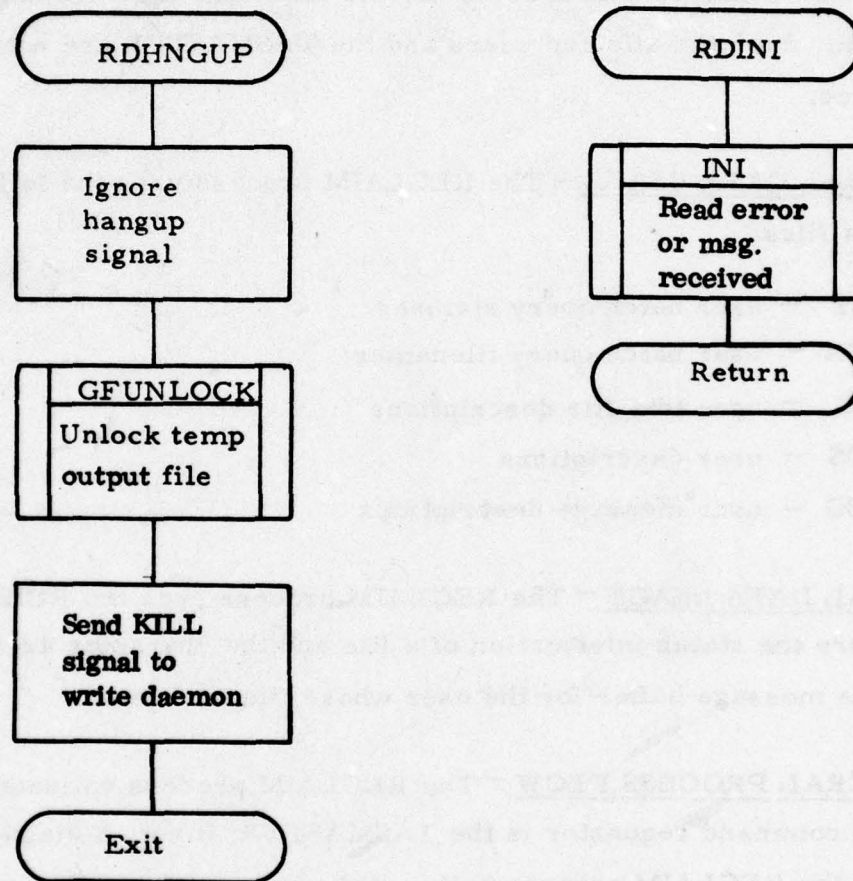


Figure 39. RD Process Data Flow (Sheet 4 of 4)

RECLAIM (File Reclamation Process)

The RECLAIM process processes the RECLAIM command. It is called by the TASMASTER to search through TAS users' edit and query/response files and to delete those files which are determined to be too old. This command aids the TASMASTER in removing old files that are burdening the TAS file system. Both the affected users and the TASMASTER are notified of the deleted files.

GLOBAL DATA USAGE - The RECLAIM process uses the following global data files:

- GBQST - user batch query statuses
- GBQFN - user batch query filenames
- GFILE - user text file descriptions
- GUIDES - user descriptions
- GMESG - user message descriptions

LOCAL DATA USAGE - The RECLAIM process uses the RCIBUF structure to store the status information of a file and the character array RCBUF to build the message buffer for the user whose file is deleted.

GENERAL PROCESS FLOW - The RECLAIM process validates that the RECLAIM command requestor is the TASMASTER; if not, a diagnostic is output and the RECLAIM process exits. Otherwise, RECLAIM saves the present time. The process next prompts the TASMASTER for the maximum number of days since the last access to an edit file.

If the input is not acceptable, the request is repeated until a satisfactory response is received. Again, the process prompts the TASMMASTER, this time for the maximum number of days since a response was delivered to a query. The input is checked and if not acceptable, the request is repeated until a satisfactory response is received.

Once the two inputs are both accepted, the user file GUDes is opened. For each user item which is non-empty, the outlined procedure is followed.

1. If GUDes has not yet been locked, global function GFLOCK locks GUDes.
2. The next GUDes item is read.
3. If the user item is empty, the next GUDes item is read.
4. If the user is logged off, RECLAIM writes out the user's logon status as being UPDATED. This prevents the user from logging on while RECLAIM is reviewing his files.
5. GUDes is unlocked and the user's file-pathnames are built.
6. Function RCFILE is called to delete old files.
7. If the user was originally logged off, the GFLOCK function locks GUDes and RECLAIM changes the user's logon status back to LOGOFF.
8. If any files of the user were deleted, a message file is written to the user notifying him of the deletions.

Once the TAS users' files have been scanned for old edit and response/ query files, the RECLAIM process terminates. (See figure 40 for data flow.)

MAJOR FUNCTION DESCRIPTIONS

RCFILE - The RCFILE function opens the specific user's GFILE and calls GFLOCK to lock GFILE. Each edit filename in GFILE has statistics retrieved for it, and if the number of days since the last access of the file is greater than the TASMMASTER's specified maximum-number-of-days-since-last-access, the following actions delete the file:

1. The edit file is unlinked.
2. The GFILE item entry is cleared and written out.
3. The edit file counter is decremented.
4. If this is the first file of the user to be deleted, a user message file is prepared; otherwise, the message file is appended to with the appropriate information.
5. A message is output to the TASMMASTER's console.

Having investigated each edit file for the user, a check is made to determine whether the file count has changed. If it has, the first item of GFILE is written out to disk. In the special case where all edit files in GFILE are deleted, GFILE is simply recreated. The global function GFUNLOCK is called to unlock GFILE.

A parallel procedure is followed for batch query and response files. RCFILE opens the given user's GBQST and calls GFLOCK to lock GBQST. Before a query file is a candidate for deletion, its status must indicate that the response to the query has been delivered and that the time since the query was responded to is greater than allowed by the TASMMASTER. If the two conditions are true, both the query and response files are unlinked, the

GBQST entry is cleared and written out, the query counter is decremented and the query's filenames are deleted from GBQFN. The user message file is created if this is the first deleted file or else is appended to with an appropriate message. The TASMASTER is informed of the "reclaimed" query/response file. After each item in GBQST is evaluated, it is determined whether a query was deleted. If so, the first item of GBQST is rewritten unless all queries in GBQST have been deleted, in which case GBQST and GBQFN is recreated. The GFUNLOCK routine unlocks GBQST and RCFILE returns.

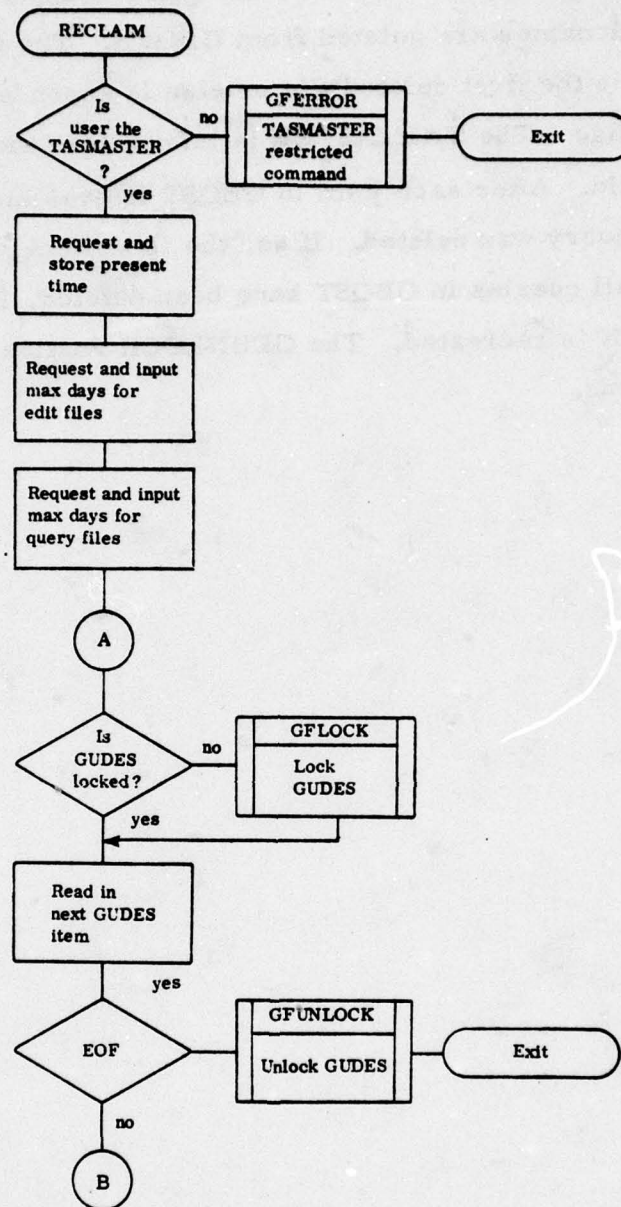


Figure 40. RECLAIM Process Data Flow (Sheet 1 of 4)

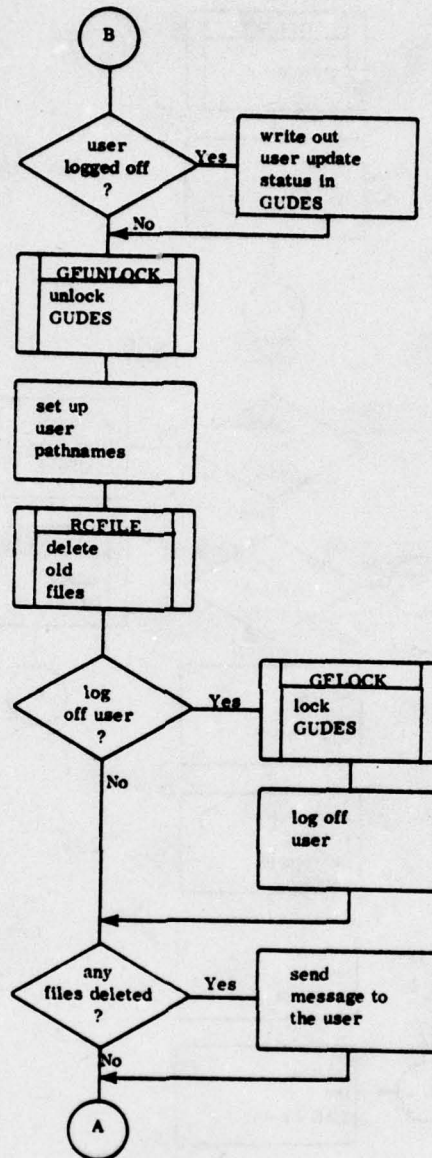


Figure 40. RECLAIM Process Data Flow (Sheet 2 of 4)

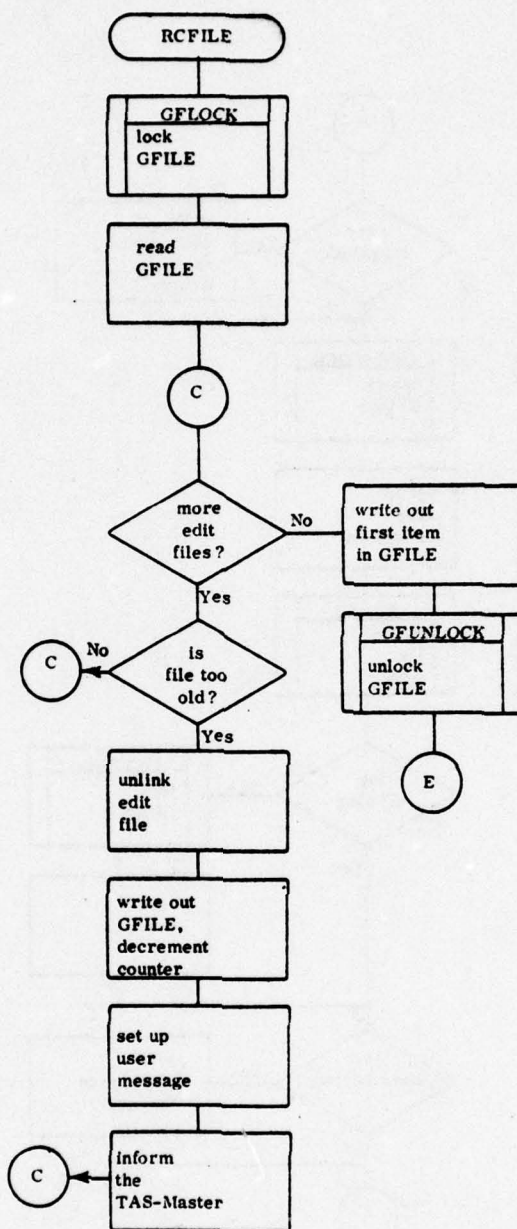


Figure 40. RECLAIM Process Data Flow (Sheet 3 of 4)

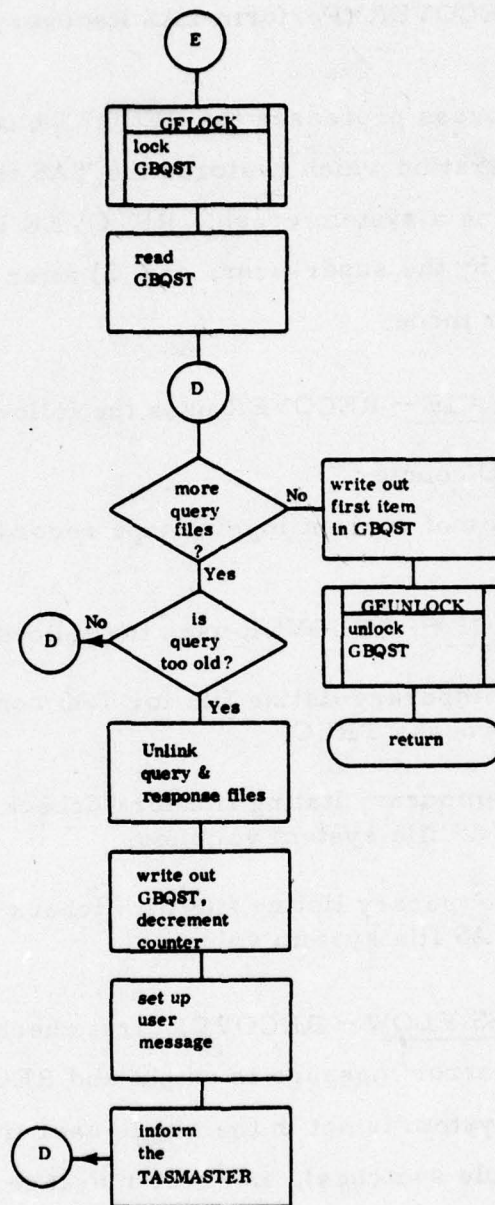


Figure 40. RECLAIM Process Data Flow (Sheet 4 of 4)

RECOVER (Perform TAS Recovery)

The RECOVER process processes the RECOVER command. It performs the recovery operation which restores the TAS file system integrity and consistency following a system crash. RECOVER is a command which may only be issued 1) by the super-user, and 2) after the system has been booted up in single user mode.

GLOBAL DATA USAGE - RECOVER uses the following global data:

GJOBID - JOBID counter
LTSIZE - number of current logging tape records

LOCAL DATA USAGE - RECOVER uses the following local data:

/tmp/clist - temporary listing file for TAS consistency check process TRCC.

/tmp/dlist - temporary listing file for "dcheck" tests of the TAS file system volumes.

/tmp/ilist - temporary listing file for "icheck" tests of the TAS file system volumes.

GENERAL PROCESS FLOW - RECOVER first checks if the user is the super-user. If not, an error message is output and RECOVER terminates. Otherwise, if the TAS system is not in the single user mode (as indicated by 00773030 in the console switches), an error message is output and RECOVER sleeps for five seconds. This cycle is repeated until the correct value is entered into the switches. RECOVER then informs the user that "icheck" and "dcheck" tests of the TAS file system volumes "/dev/rhp00, 1, 2, 3" are being run. The processes ICHECK and DCHECK are called to

perform the actual tests. If any errors are encountered, the user is given the option of reloading the TAS file system from backup magnetic tape or continuing. If he so desires, RECOVER executes the TASLOAD process to perform the recovery from tape, then outputs an end message and terminates. If no errors were encountered from the "icheck" and "dcheck" tests, or the user chooses to continue, the user is informed that consistency tests are being run on the TAS file system. RECOVER then calls via a fork/execute sequence the process TRCC to perform the actual consistency check of the TAS data files. If fatal errors were encountered in the consistency check of the TAS file system, an introductory message is output followed by a listing of all error/modifications found/made to the file system. The temporary listing file "/tmp/clist" is unlinked and the user is given the option of reloading the TAS file system from backup magnetic tape. If he so desires, RECOVER executes the command process TASLOAD to perform the magnetic tape recovery. An end message is then output and RECOVER terminates. If no fatal errors were encountered, but some modifications were made to the TAS file system, the listing file "/tmp/clist" is output to the user. Then, as in the case of no consistency errors, the temporary listing file "/tmp/clist" is unlinked, GJOBID is opened and read, the last TAS JOBID used is output to the user, the contents of LTSIZE (number of physical COINI logging tape records) is output, an end message is displayed, and RECOVER terminates. See figure 41 for data flow.

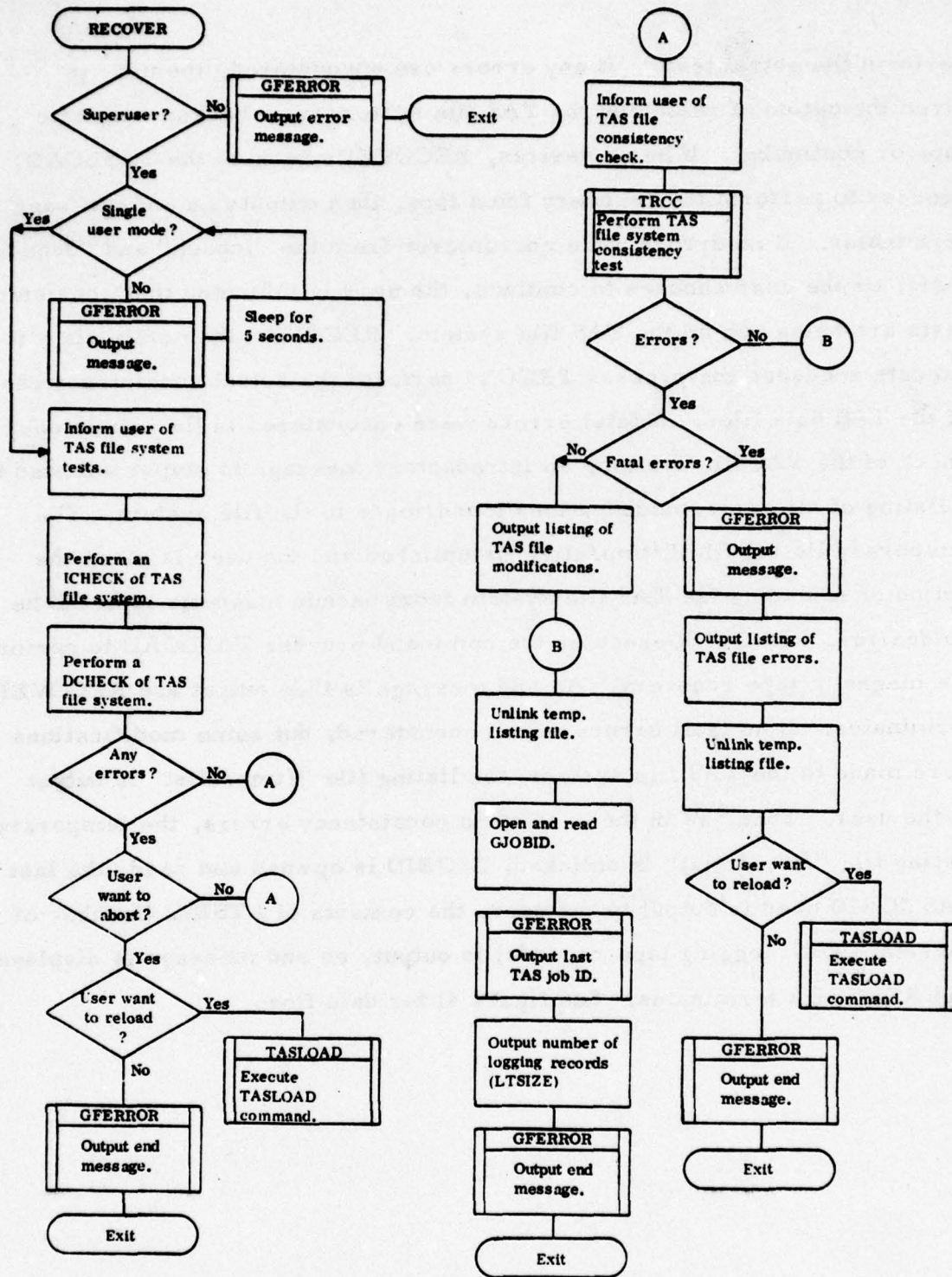


Figure 41. RECOVER Process Data Flow

TRCC (Perform a Consistency Check of the TAS File System Data)

TRCC is the process which is responsible for performing a consistency check of the data within the TAS file system. The results of the analysis are given in a report.

GLOBAL DATA USAGE - TRCC uses the following global data:

GAPPL - application authorization data
GAUTH - user authorization data
GBQST - user batch query statuses
GFILE - user text file descriptions
GFILS - file authorization data
GILOG - interactive application logon information
GJOBS - BQRD batch job descriptions
GMESG - user message descriptions
GPROJ - project descriptions
GRES P - batch query response processor pathnames
GUDS - user descriptions

LOCAL DATA USAGE - TRCC uses the following local data:

TREC Error count detected by TRCC
TR_GLK Last of potential locked files

GENERAL PROCESS FLOW - TRCC first checks for an input argument.

If one is present, the specified file is created and the standard output is

redirected to that file. A report header is then output giving the date and time of the consistency check.

Pass I of the TAS file system consistency check begins with the removal of all global lock files. For each global file listed in the structure TR-GLK, the function TR_UNLOCK is called to unlock to file if it is locked.

When all global files have been unlocked, the function TR-2 is called to make the second pass of the TAS file system consistency check (validation of user files). When that pass is completed, TRCC2 is executed and passed the error count of the first two passes to complete the consistency check of the TAS file system. See figure 42 for data flow.

MAJOR FUNCTION DESCRIPTIONS

TR-2 (Validate TAS User Files) - When called, TR-2 first opens GUDS and seeks to item one. For each item in GUDS an update flag is cleared, and if the item is occupied, TR-USER is called to validate the files for that user. If, after processing the user, the GUDS item was modified, the item is written and the next one is processed.

At end-of-file, GUDS is closed and TR-2 returns.

TR-USER (Validate TAS Files for a Specific User) - When called, TR-USER writes the user's sni to the output file and checks if the user was being created or deleted by AAFM. If so, a message is output, AARMVSN is called via a fork/execute sequence to remove the user from TAS. TR-USER waits for completion of the process and returns.

If the user is not being created or deleted, but is being modified, the function TR-PSWD is called to validate the user's "/etc/passwd" entry.

If the entry is not valid, an error message is output and TRCC terminates. Otherwise, trGAUTH is called to validate the user's GAUTH (GPROJ, GAPPL, GFILS) files.

If the user is not logged off TAS, the GUDS logon status is set to logged off and the GUDS update flag is set. Finally, the functions trGFILE, trGBQST, and trGMESG are called to validate the user's GFILE, GBQST and GMESG files, respectively. TR-USER then returns.

TR-PSWD (Validate a User's Password File Entry) - When called, TR-PSWD opens the password file "/etc/passwd" and searches each line in the file for a match to the input user ID. If the name in the line is eight characters long and the user ID matches, the user sni is tested. If the sni does not match, a message is output, the GUDS sni field is changed to match the "passwd" file's name and the GUDS update flag is set.

If a match was found to the user ID, a "match found" flag is set, and the search terminates. Otherwise, the search terminates with the end-of-file on "/etc/passwd". At this point the password file is closed, and if a match was found, 0 is returned; otherwise -1 is returned.

trGAUTH (Validate a User's GAUTH, GPROJ, GAPPL, GFILS Files) - When called, trGAUTH builds the GAUTH file pathname for the current user, opens GAUTH and reads it. If the GAUTH and GUDS snis differ, the GAUTH sni field is altered to match GUDS, a message is output, and a GAUTH update flag is set.

The number of compartment names for the user is then counted, and if this does not agree with the number of names stored in GAUTH, that number is reset, a message is output and the GAUTH update flag is set.

The function trGPROJ is then called to validate the user's GPROJ (GAPPL, GFILS) files. When this is completed, if GAUTH has been modified, the user's GAUTH file is written. GAUTH is then closed and trGAUTH returns.

trGPROJ (Validate a User's GPROJ, GAPPL, GFILS Files) - trGPROJ builds the GPROJ file pathname for the current user, opens it and initializes the project count.

For each item in the user's GPROJ, if the item is occupied, that project name is output and the project directory pathname is built. If there are no applications listed in the GPROJ item for this project and it is not the default project (zero), a message is output, the project directory (if any) is unlinked, the current GPROJ item is cleared and the GPROJ item update flag is set.

If there are applications for this project, but the project directory does not exist, a message is output, the GPROJ item is cleared and its update flag is set.

If there are applications for this project and the project directory exists, the GAPPL file pathname for this user-project is built. If the GAPPL file exists, trGAPPL is called to validate the GAPPL, GFILS data; otherwise, a message is output, the project directory is unlinked, the GPROJ item is cleared and its update flag is set.

At this point, if the GPROJ item has been modified, the item is written and if the item is still occupied, the current project is counted.

After processing all GPROJ items, the number of projects counted is compared with the number in GAUTH. If these numbers differ, a message

is output, GAUTH is modified to reflect the correct number and the GAUTH update flag is set. Finally, GPROJ is closed and trGPROJ returns.

trGAPPL (Validate a User-project's GAPPL, GFILS Files) - trGAPPL opens the GAPPL file for the current user-project (the pathname is built by trGPROJ) and initializes the application count.

Each item of GAPPL is read and if the item is occupied, the host name is output and it is determined if the host is batch or interactive. If the host is interactive, the GILOG pathname is built, and if the file does not exist, a message is output, the current GAPPL item is cleared and its update flag is set. If the host is batch, its GFILS file pathname is built, and if its GFILS exists, trGFILS is called to validate it; otherwise a message is output, the GAPPL item is cleared and its update flag is set.

If the GAPPL item was occupied, and it has been modified, the item is written. Finally, if the item is still occupied the application is counted before reading and processing the next item.

When all GAPPL items have been processed, a test is made to determine if the last application has been removed. If so, a message is output, GAPPL and its project directory are unlinked, the current GPROJ item is cleared and its update flag is set.

If some applications still remain, the number of applications counted is compared to the number in the current GPROJ item. If these numbers differ, the GPROJ item is modified to reflect the correct number and its update flag is set.

Finally, GAPPL is closed and trGAPPL returns.

trGFILS (Validate a User-project-application's GFILS File) – trGFILS opens GFILS and zeroes the host-file count. Then for each occupied item in GFILS, the host-filename is output and counted.

When all GFILS items have been processed, if no files were counted, a message is output, GFILS is unlinked, the current GAPPL item is cleared and its update flag is set.

Otherwise, if some files were counted, but the number of files in the current GAPPL item differs from the number counted, a message is output, the current GAPPL item is modified to reflect the correct number of host-files and its update flag is set. GFILS is then closed and trGFILS returns.

trGBQST (Validate a User's GBQST File) – trGBQST builds the GBQST file pathname for the current user and calls TR_UNLOCK to unlock GBQST.

If GBQST was locked, it is opened, item zero is read and the query counter is initialized. For each occupied item of GBQST, an update flag is cleared and the query is counted. If the job status is answered or aborted, the response filename is built, and if the file exists, the function TR-R-JOB is called to remove that job's entries from GJOBS and GRESP. Otherwise, a message is output, the job status is set to pending in the current GBQST item and its update flag is set.

If the update flag is set for the current GBQST item, that item is written.

When all GBQST items have been processed, the number of queries counted is compared to the number of queries in the GBQST item zero. If they differ, a message is output, GBQST item zero is modified to reflect the correct number of queries and the item is written.

Finally, GBQST is closed, and trGBQST returns.

TR-R-JOB (Remove a Job's Entries from GJOBS and GRESP) - TR-R-JOB builds the GJOBS pathname for the current host and opens the file. GJOBS is then searched for a match to the input JOBID.

When a match is found, a message is output, the GJOBS data file is unlinked, TR-R-RSP is called to remove the job's response processor pathname from GRESP, the GJOBS item is cleared and the GJOBS item is written.

GJOBS is then closed and TR-R-JOB returns.

trGFILE (Validate a User's Text Files) - trGFILE first builds the GFILE file pathname and calls TR_UNLOCK to unlock the file.

If GFILE was not locked, trGFILE returns. Otherwise, the file is opened and its item zero is read. If GFILE has zero length, the file is closed and trGFILE returns. Otherwise, the number of text files is initialized.

For every occupied item in GFILE, if its text file exists, it is counted; otherwise a message is output, the current GFILE item is cleared and the item is written.

When all GFILE items have been processed, if the number of text files counted differs from the number of text files in GFILE item zero, GFILE item zero is modified to reflect the correct number of text files and item zero is written.

Finally, GFILE is closed and trGFILE returns.

trGMESG (Validate a User's Messages) - trGMESG first builds the GMESG file pathname and calls TR_UNLOCK to unlock GMESG.

If GMESG was not locked, trGMESG returns. Otherwise, the file is opened and its item zero is read. If GMESG has zero length, the file is closed and trGMESG returns. Otherwise, the number of messages is initialized.

For every occupied item in GMESG, if its message file exists, it is counted; otherwise a message is output, the current GMESG item is cleared and the item is written.

When all GMESG items have been processed, if the number of messages counted differs from the number of messages in GMESG item zero, GMESG item zero is modified to reflect the correct number of message files and item zero is written.

Finally, GMESG is closed and trGMESG returns.

TR_UNLOCK (Unlock a File) - TR_UNLOCK first builds the pathname of the lock file and unlinks the file. If the unlink is successful, (i. e., the file was locked) -1 is returned; otherwise 0 is returned.

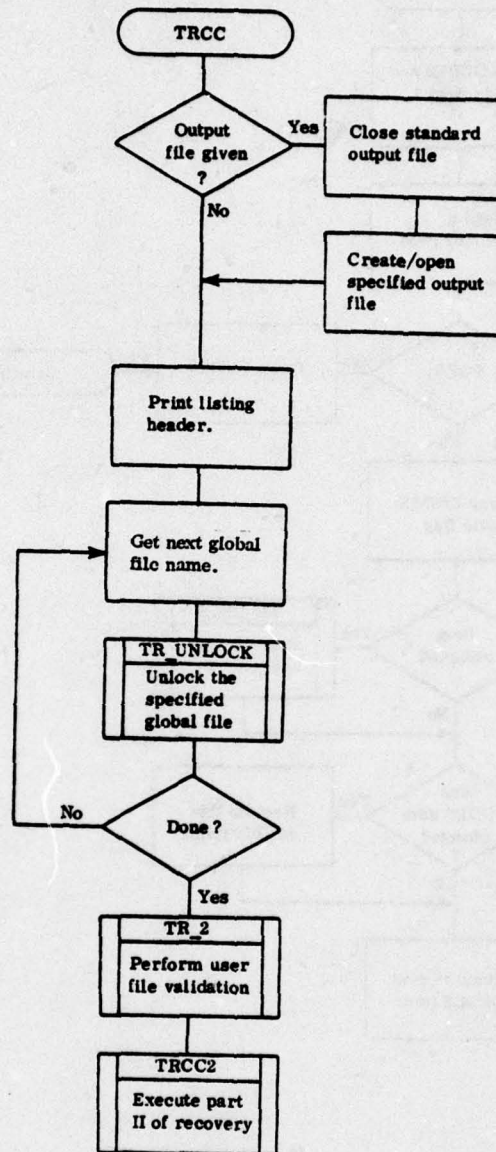


Figure 42. TRCC Process Data Flow (Sheet 1 of 13)

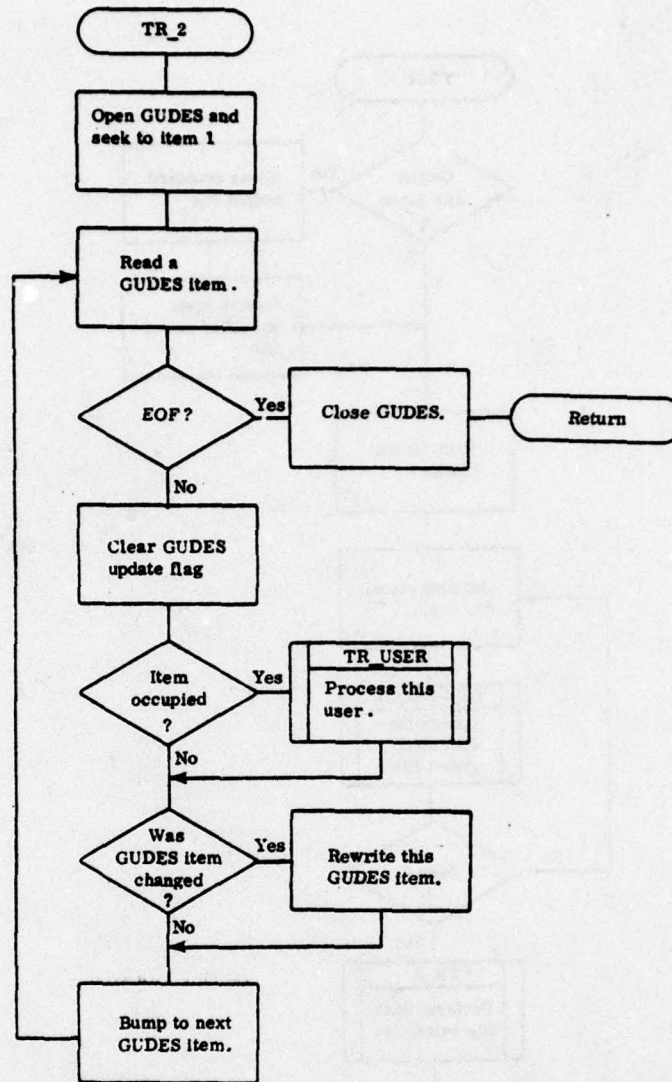


Figure 42. TRCC Process Data Flow (Sheet 2 of 13)

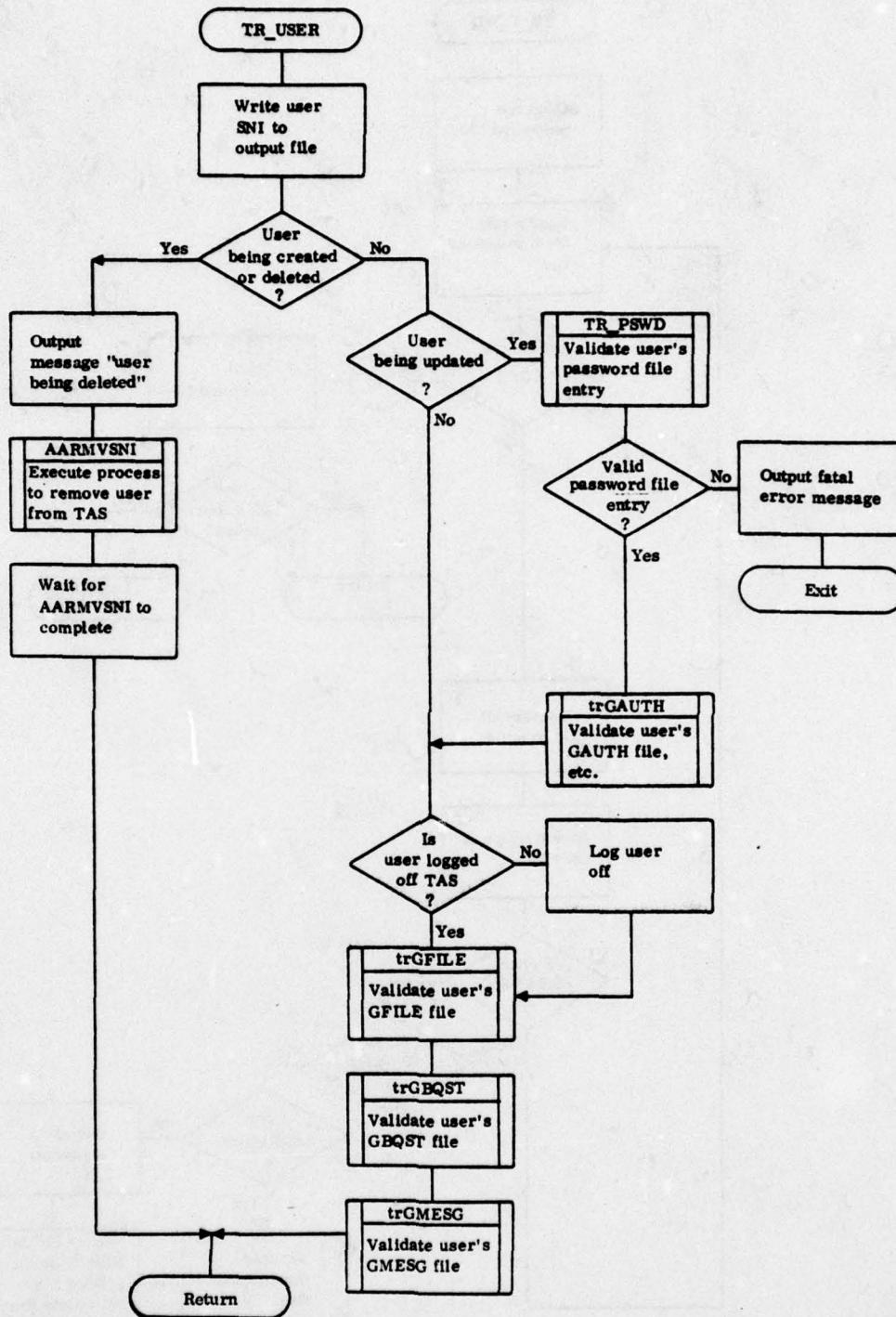


Figure 42. TRCC Process Data Flow (Sheet 3 of 13)

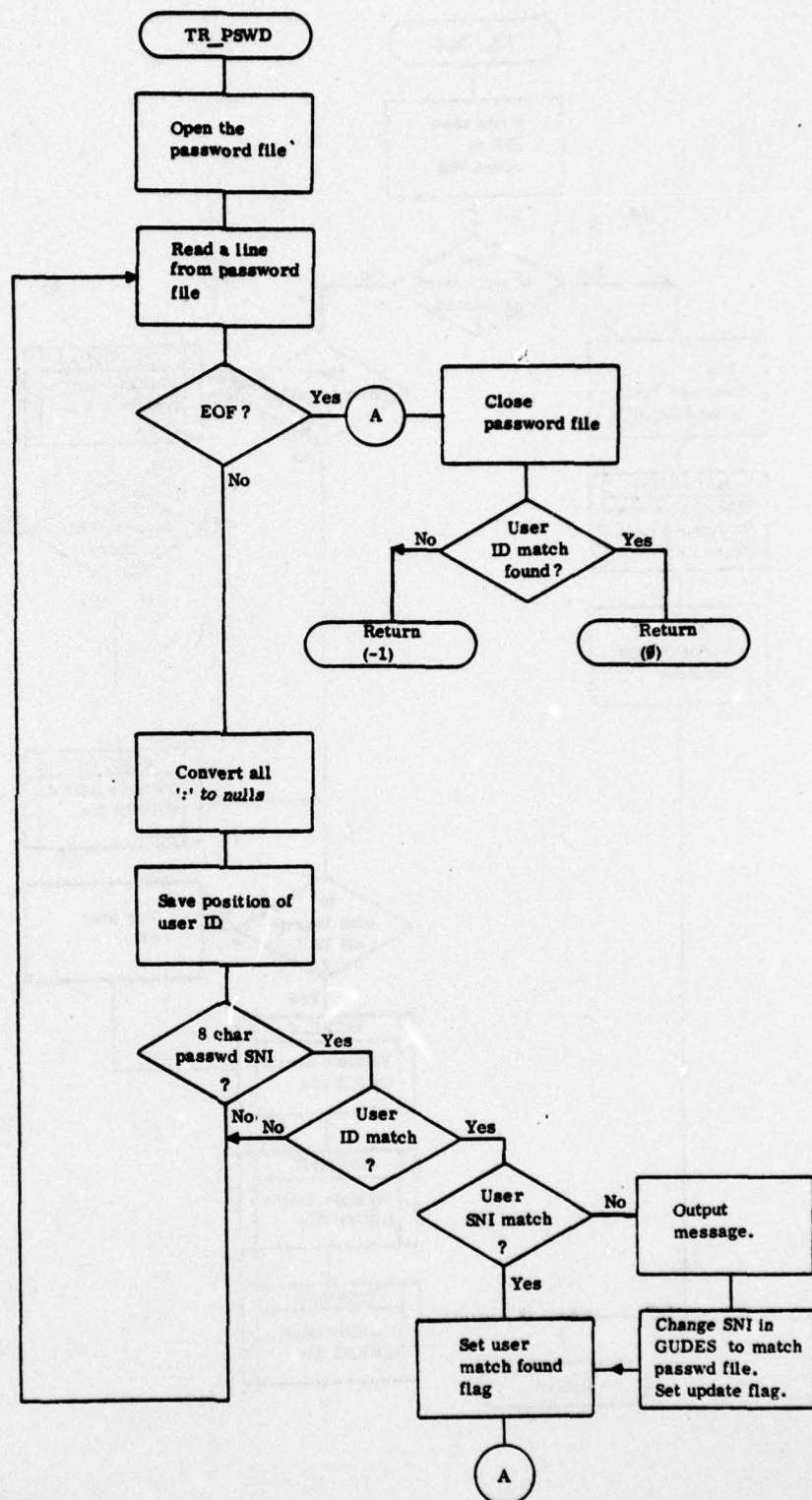


Figure 42. TRCC Process Data Flow (Sheet 4 of 13)

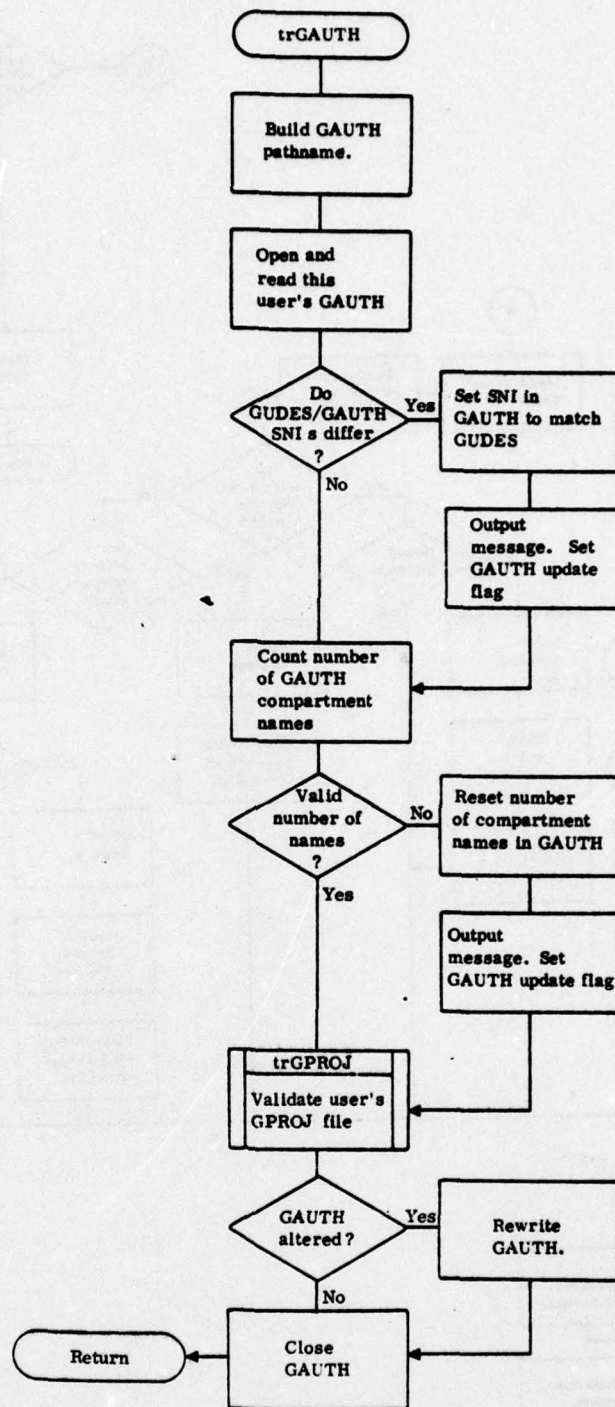


Figure 42. TRCC Process Data Flow (Sheet 5 of 13)

Figure 42. TRCC Process Data Flow (Sheet 6 of 13)

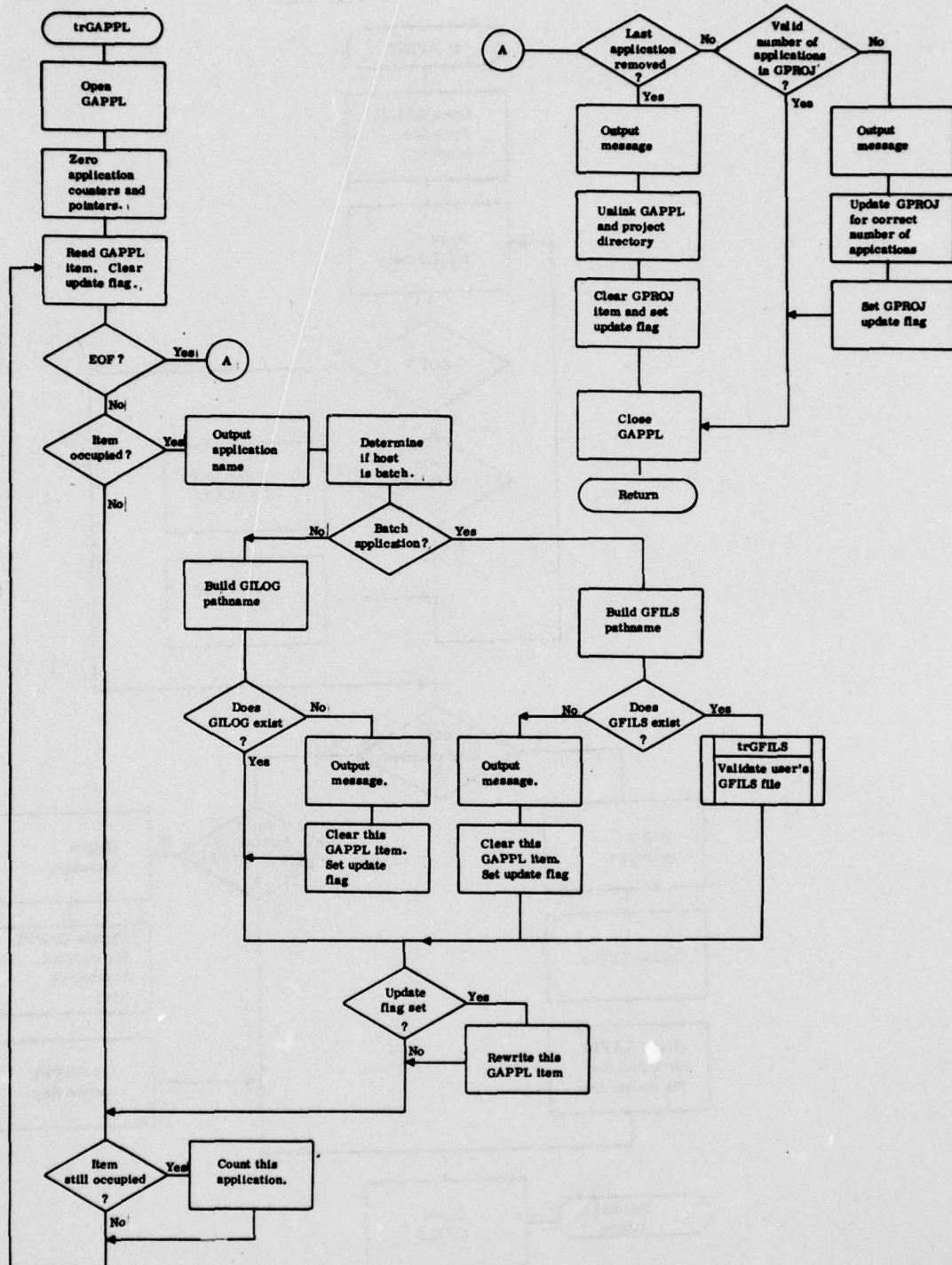


Figure 42. TRCC Process Data Flow (Sheet 7 of 13)

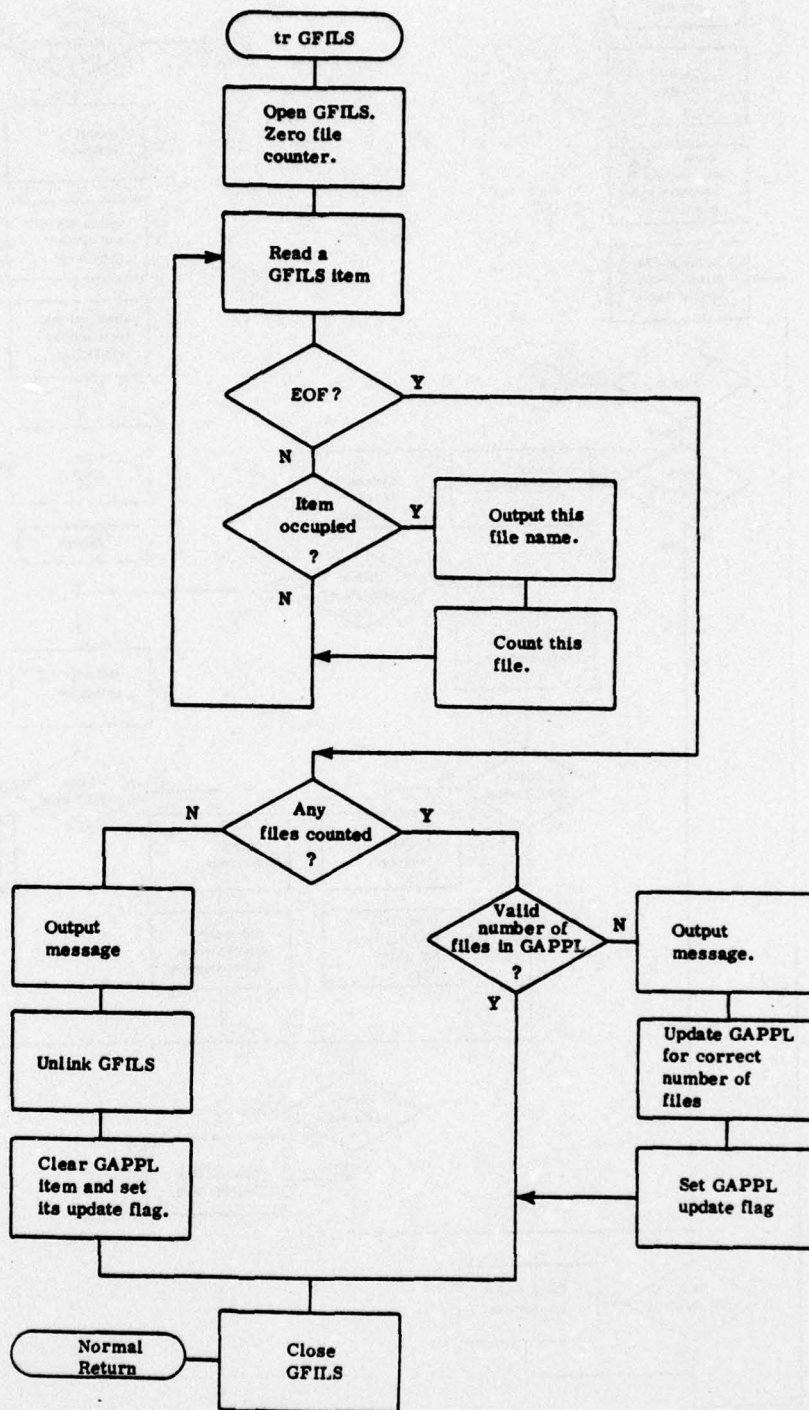


Figure 42. TRCC Process Data Flow (Sheet 8 of 13)

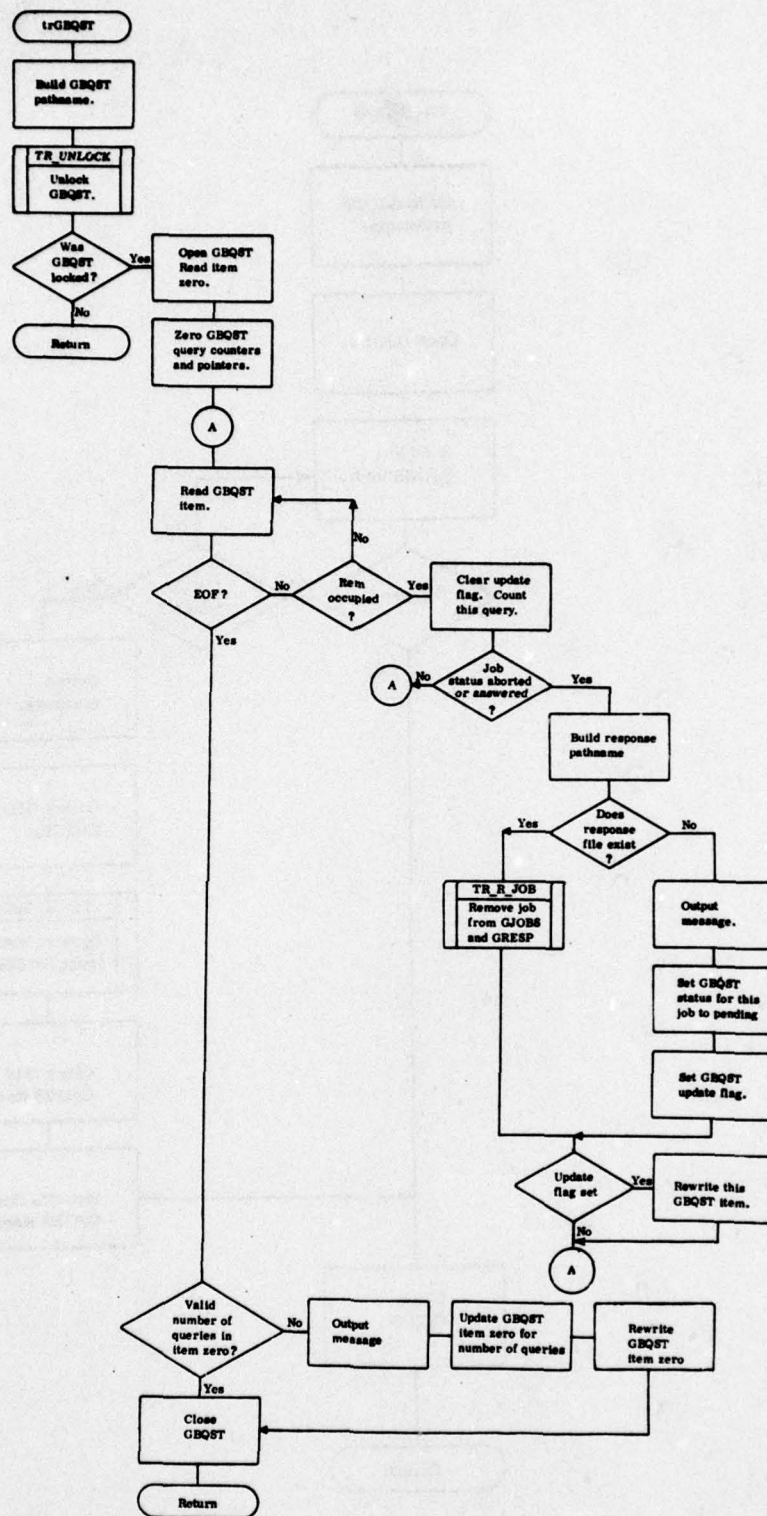


Figure 42. TRCC Process Data Flow (Sheet 9 of 13)

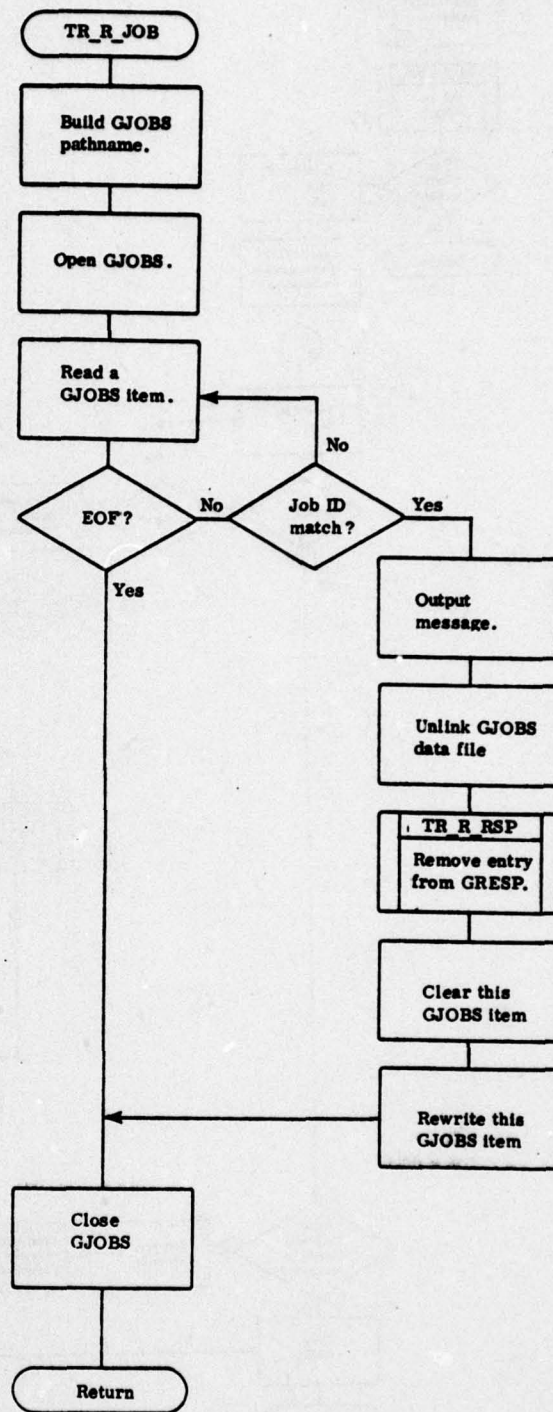


Figure 42. TRCC Process Data Flow (Sheet 10 of 13)

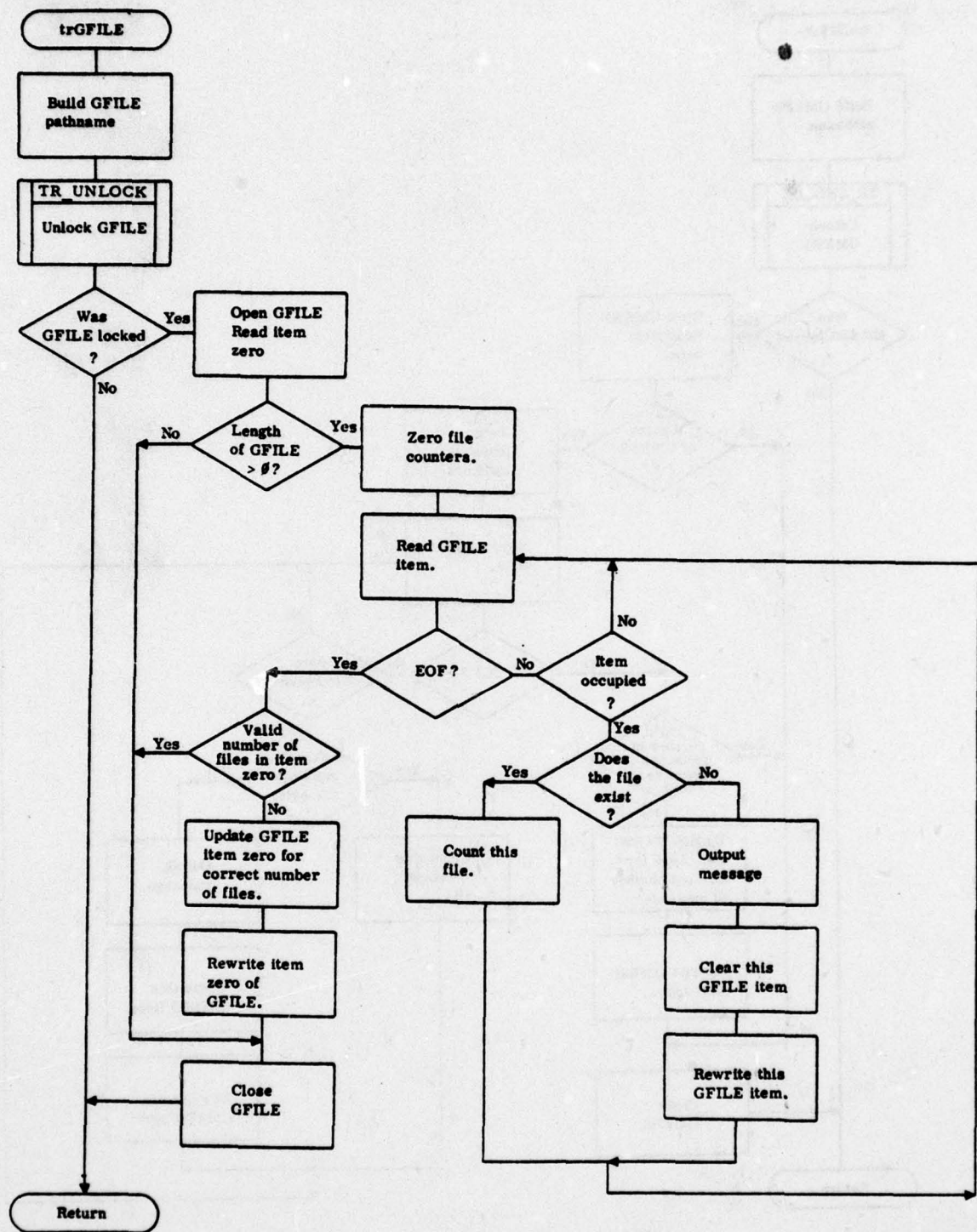


Figure 42. TRCC Process Data Flow (Sheet 11 of 13)

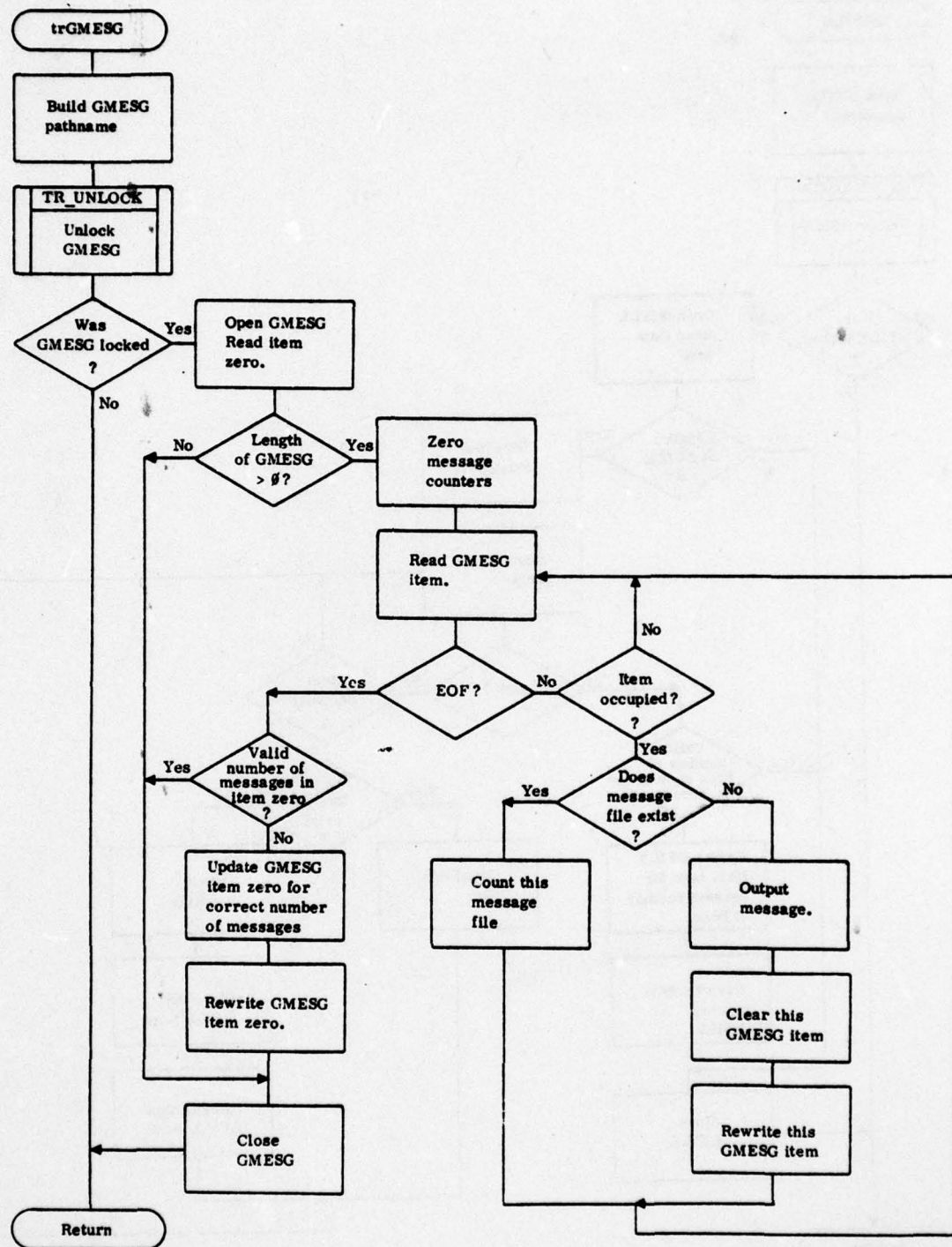


Figure 42. TRCC Process Data Flow (Sheet 12 of 13)

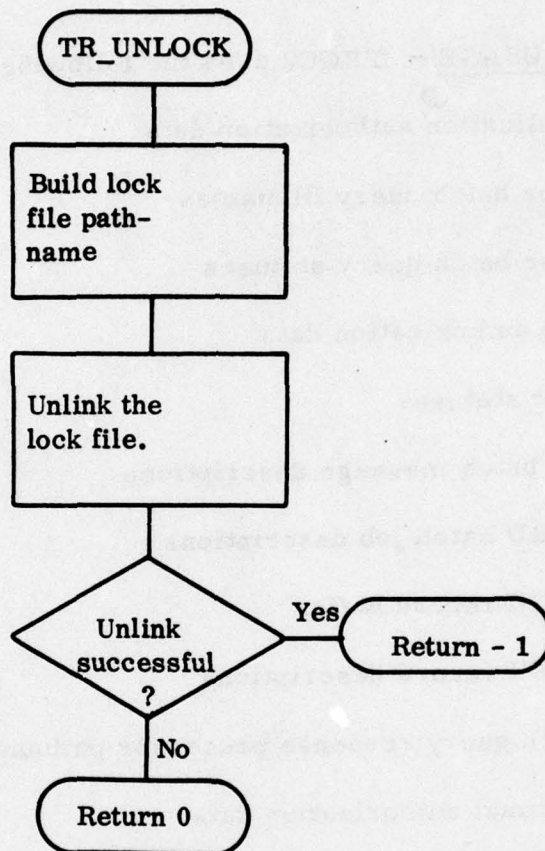


Figure 42. TRCC Process Data Flow (Sheet 13 of 13)

AD-A050 967

LOGICON INC SAN DIEGO CALIF

F/G 9/2

TERMINAL ACCESS SYSTEM (TAS) SYSTEM DESIGN SPECIFICATION.(U)

JAN 78 R M BARNHART, L R ERICKSON

N00014-76-C-0899

UNCLASSIFIED

76-C-0899-6

NL

5 OF 8

AD
A060 967



TRCC2 (Part II of the TAS File System Data Consistency Check)

TRCC2 is the process which is responsible for consistency check passes 3, 4, and 5 of the TAS file system data.

GLOBAL DATA USAGE - TRCC2 uses the following global data:

GAPPL	-	application authorization data
GBQFN	-	user batch query filenames
GBQST	-	user batch query statuses
GFILS	-	file authorization data
GHOST	-	host statuses
GIMSG	-	INI batch message descriptions
GJOBS	-	BQRD batch job descriptions
GLBUF	-	COINI record buffer
GLJOBS	-	COINI record descriptions
GRES P	-	batch query response processor pathnames
GTAUT	-	terminal authorization data
GTERM	-	terminal descriptions
GUIDES	-	user descriptions

LOCAL DATA USAGE - TRCC2 uses the following local data:

TREC	Error count detected by TRCC2
TR-HOST	Host ID

GENERAL PROCESS FLOW - TRCC2 begins with pass 3 (TR-3) over the TAS file system, validation of TAS terminal data. First the error count from TRCC is picked up from the input argument and saved. Then GTERM is opened and each item read and processed.

For each GTERM item an update flag is cleared, and if the item is occupied, the terminal name is output and the terminal is processed. If the terminal is being created or deleted by AAFM, a message is output and TR-R-TERM is called to remove the terminal from TAS.

If the terminal is not being created or deleted, but is being modified, trGTAUT is called to process the terminal's GTAUT (GAPPL, GFILS) files. If the terminal is not logged off, the GTERM terminal status is set to "logged off" and the GTERM item update flag is set. If the GTERM item has been modified, the item is written.

When all GTERM items have been processed, GTERM is closed, and functions TR-4 and TR-5 are called to validate TAS job files and TAS logging files respectively. TRCC2 then exits with the error count as status. See figure 43 for data flow.

MAJOR FUNCTION DESCRIPTIONS

TR-R-TERM (Remove a Terminal from TAS) - TR-R-TERM first unlinks the specified terminal's GTAUT, GAPPL and GFILS files. GTERM is then opened and searched for a match on the terminal ID. When it is found, that GTERM item is cleared and rewritten. GTERM is then closed and TR-R-TERM returns.

trGTAUT (Validate a Terminal's GTAUT, GAPPL and GFILS Files) - trGTAUT first builds the GTAUT file pathname for the current terminal, then opens and reads the file and clears its update flag. The items in the

terminal's compartment name list are then processed and each occupied item is counted. If the number of compartment names counted differs from the number in the GTAUT file, a message is output, the GTAUT file is modified to reflect the correct number of compartments and its update flag is set. Then trTAPPL is called to validate the terminal's GAPPL, GFILS files. When this is completed, if the GTAUT update flag is set, the file is written. GTAUT is then closed and trGTAUT returns.

trTAPPL (Validate a Terminal's GAPPL, GFILS Files) - trTAPPL opens the current terminal's GAPPL file and initializes the application counters. Next, each GAPPL item is read and processed.

For each GAPPL item an update flag is first cleared. Then if the item is occupied, the application name is output and it is determined whether the host is batch or interactive. If the host is batch, the GFILS file pathname is built for the current terminal-host. If GFILS exists, trTFILS is called to validate it. Otherwise, a message is output, the current GAPPL item is cleared and its update flag is set.

If the update flag for an item is set, the item is written. Finally, if the item is still occupied, it is counted before the next item is processed.

When all GAPPL items have been processed, if the number of applications counted differs from the number in the terminal's GTAUT file, GTAUT is modified to reflect the correct number and its update flag is set.

Finally, GAPPL is closed and trTAPPL returns.

trTFILS (Validate a Terminal-host's GFILS File) – trTFILS opens the terminal-host's GFILS and initializes the host-file count. Next, for each occupied item of GFILS, the host-file name is output and the file is counted.

When all GFILS items have been processed, if no host-files have been counted, a message is output, the terminal's GFILS file is unlinked, the current GAPPL item is cleared and its update flag is set.

Otherwise, if the number of host-files counted differs from the number in the current GAPPL item, a message is output, the GAPPL item is modified to reflect the correct number and its update flag is set.

Finally, GFILS is closed and trTFILS returns.

TR-4 (Validate TAS Batch Job Files) – TR-4 first opens GHOST. For each batch host item in GHOST, the application name referred to is output and the function trGJOBS is called to process the GJOBS file for that host.

When all items of GHOST have been processed, GHOST is closed, trGIMSG is called to process the GIMSG file and TR-4 returns.

trGIMSG (Process the GIMSG File) – When called by TR-4, trGIMSG first opens the GIMSG file. Then for each occupied item in GIMSG, the INI data file for the referenced job is unlinked.

When all GIMSG items have been processed, GIMSG is closed and recreated and trGIMSG returns.

trGJOBS (Process a Host's GJOBS File) – When called by TR-4, trGJOBS first builds the GJOBS file pathname for the current host, opens the file and unlocks it. Then each item of GJOBS is read and processed.

For each GJOBS item which is occupied and for which the job status is pending or transmitted, the BQRD data file pathname is built. If this file does not exist, and the job is an INTG, SRVC (TRACE) or SRVC (PRINT), a message is output, TR-R-LOG is called to remove the TAS logging file for the specified job, the current GJOBS item is cleared and that item is written.

If the BQRD data file does exist, and the referenced job is an INTG, the query file pathname is built. If the query file does not exist, a message is output and TR-R-TASJOB is called to remove the specified job from TAS.

When all job items in GJOBS have been processed, GJOBS is closed and trGJOBS returns.

TR-R-TASJOB (Remove a Job from the TAS Environment) - TR-R-TASJOB builds the BQRD data file pathname and unlinks the file. TR-R-RSP is then called to remove the response processor pathname from GRESP. If the job was a user's INTG, TR-R-BQST is called to remove the user's GBQST entry for this job. All TAS logging file and directory entries for the job are removed, the current GJOBS item is cleared and rewritten before TR-R-TASJOB returns.

TR-R-BQST (Remove a User's GBQST Entry for a Specified Job) - TR-R-BQST first builds the GBQST file pathname for the specified user, opens the file and reads its item zero. If the file has non-zero length, a search is made for a match on the specified JOBID. If a match is found, the job count in item zero is decremented, and if it is then zero, GBQST and GBQFN are recreated; otherwise, TR-C-BQFN is called to remove the GBQFN entries for this job, the current GBQST item is cleared and rewritten and GBQST item zero is rewritten.

Finally, GBQST is closed before TR-R-BQST returns.

TR-C-BQFN (Clear the Entries for a Specified Job from GBQFN) -

TR-C-BQFN first builds the GBQFN file pathname, opens the file and reads its item zero. The file is then positioned to the first item for the specified job. This and the successive items containing host-file names for the specified query are cleared and rewritten. If these items were at the end of GBQFN, the pointer to the next available item (in item zero) is decremented and item zero is rewritten. Finally, GBQFN is closed before TR-C-BQFN returns.

TR-R-RSP (Remove the Response Processor Pathname from GRESP) -

TR-R-RSP first builds the GRESP pathname and opens the file. The desired block number containing the response processor pathname to be deleted is calculated from the absolute pointer, and similarly the displacement within that block of the pathname. The desired block is then read, and the pathname characters (including the terminating null) are set to -1's. The block is then rewritten and GRESP is closed before TR-R-RSP returns.

TR-5 (Validate TAS Logging Files) - TR-5 simply calls trGLBUF to validate the COINI record logging tape buffer and trGLJOBS to validate the COINI logging record directory.

trGLBUF (Validate the COINI Record Tape Buffer) - trGLBUF first unlocks GLBUF (if necessary), then opens the file, reads its item zero (index record) and clears the GLBUF update flag.

For each of the (up to 10) COINI record JOBIDs and full records which should be present in GLBUF (as indicated by the number of records field in item zero), trGLBUF verifies that a COINI record JOBID is in its slot

in the index item and that the COINI record itself is in its corresponding item in GLBUF.

If a JOBID should fail to be in its slot, a message is output, the number of records is set to the number processed successfully so far, the GLBUF update flag is set and the validation loop is terminated.

If a COINI record should fail to exist in its indicated item within GLBUF, a message is output, the JOBID slot for this record within item zero is cleared, the number of records is set to the number processed successfully so far, the GLBUF update flag is set and the validation loop is terminated.

When the validation loop is completed, if the GLBUF update flag is set, a message is output and GLBUF is rewritten. Finally, GLBUF is closed and trGLBUF returns.

trGLJOBS (Validate the COINI Record Logging File) - trGLJOBS first unlocks GLJOBS (if necessary), opens the file and zeroes the job counter.

Next, for each occupied item in GLJOBS, a COINI record pathname is built for the indicated job. If the COINI record file does not exist, a message is output and the current GLJOBS item is cleared and rewritten. If the COINI record file does exist, but is for an interactive session rather than for a batch query, a message is output, the COINI record file is unlinked and the current GLJOBS item is cleared and rewritten.

When all GLJOBS items have been processed, GLJOBS is closed and trGLJOBS returns.

TR_UNLOCK (Unlock a File) - TR_UNLOCK first builds the pathname of the lock file and unlinks the file. If the unlink is successful, (i. e., the file was locked) - 1 is returned; otherwise 0 is returned.

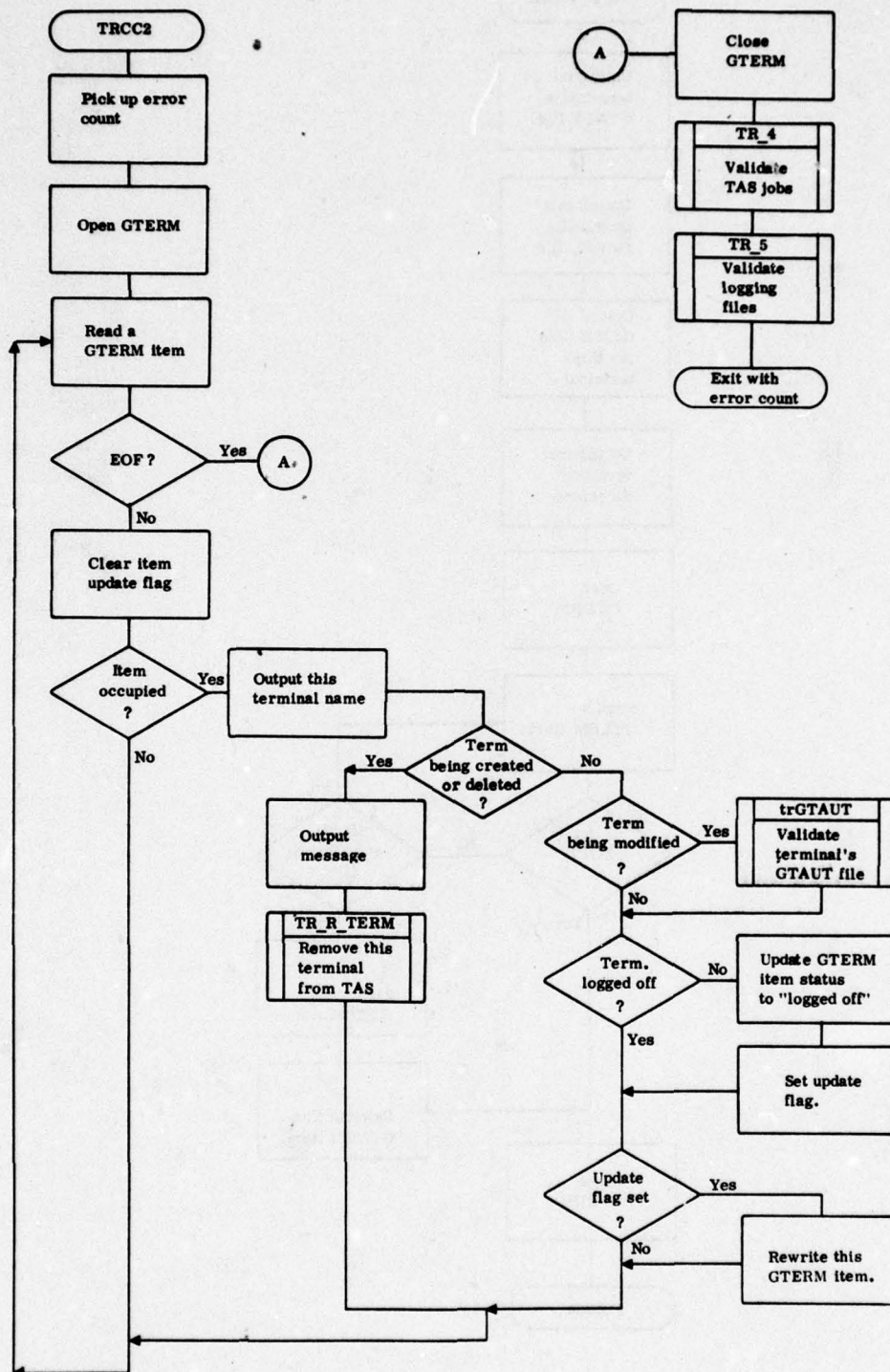


Figure 43. TRCC2 Process Data Flow (Sheet 1 of 16)

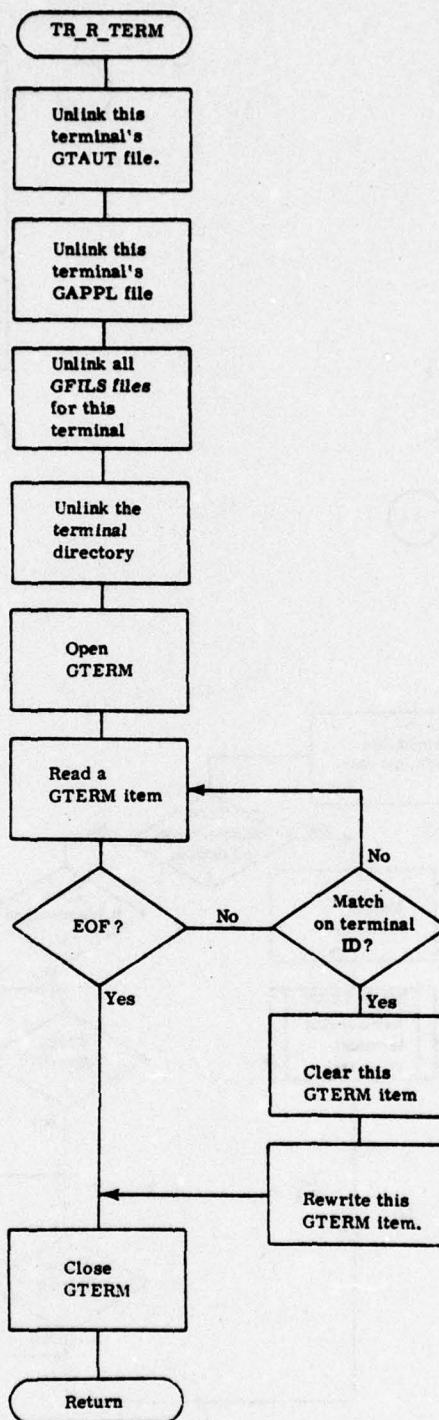


Figure 43. TRCC2 Process Data Flow (Sheet 2 of 16)

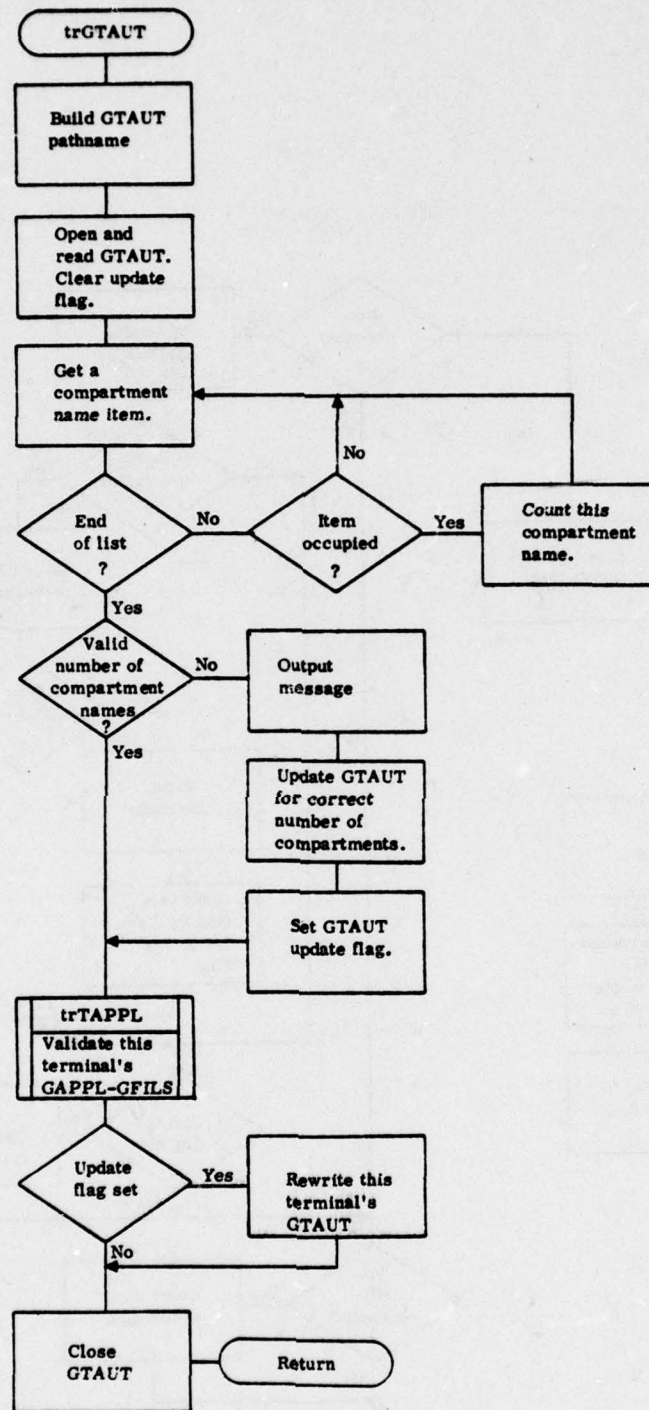


Figure 43. TRCC2 Process Data Flow (Sheet 3 of 16)

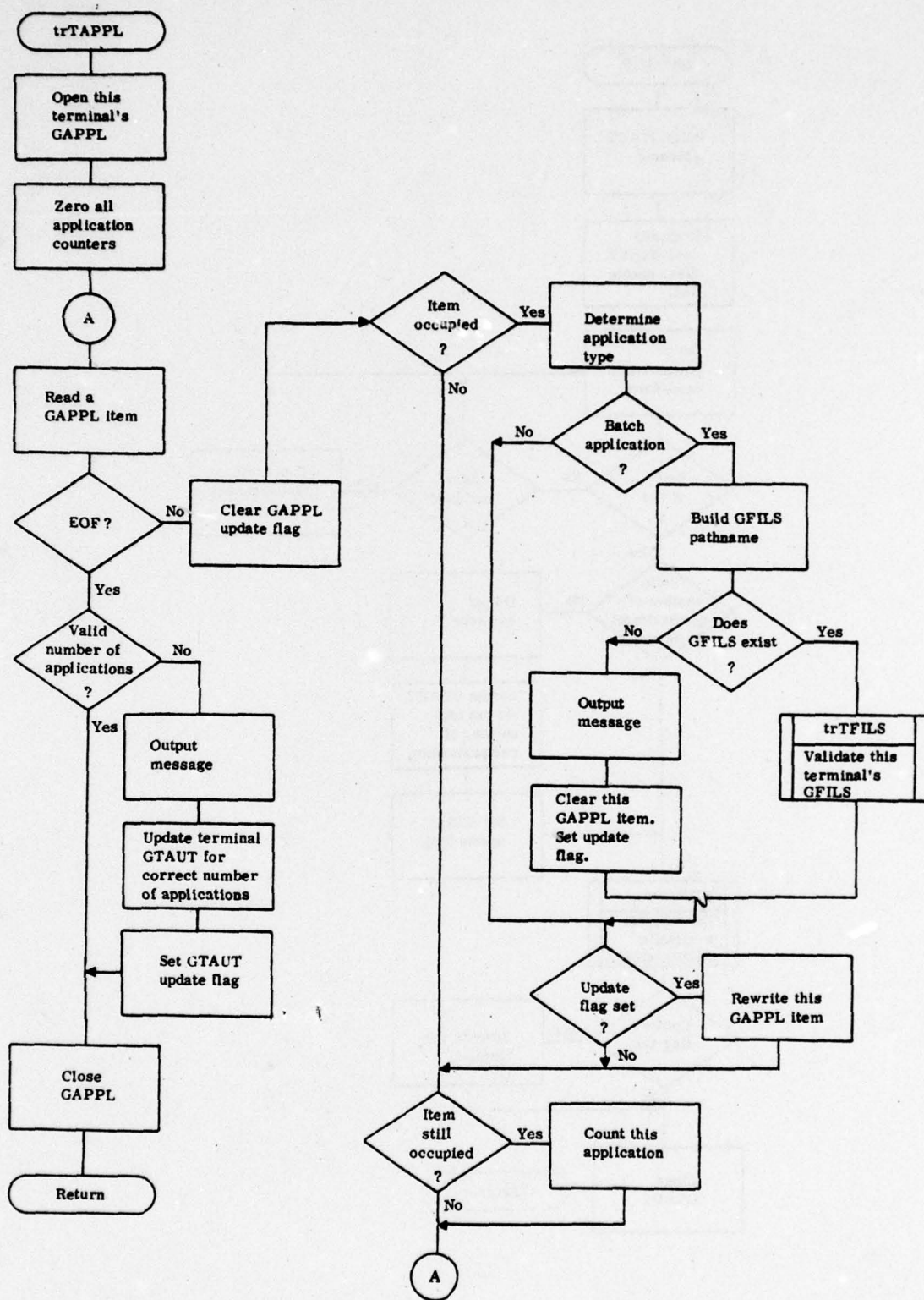


Figure 43. TRCC2 Process Data Flow (Sheet 4 of 16)

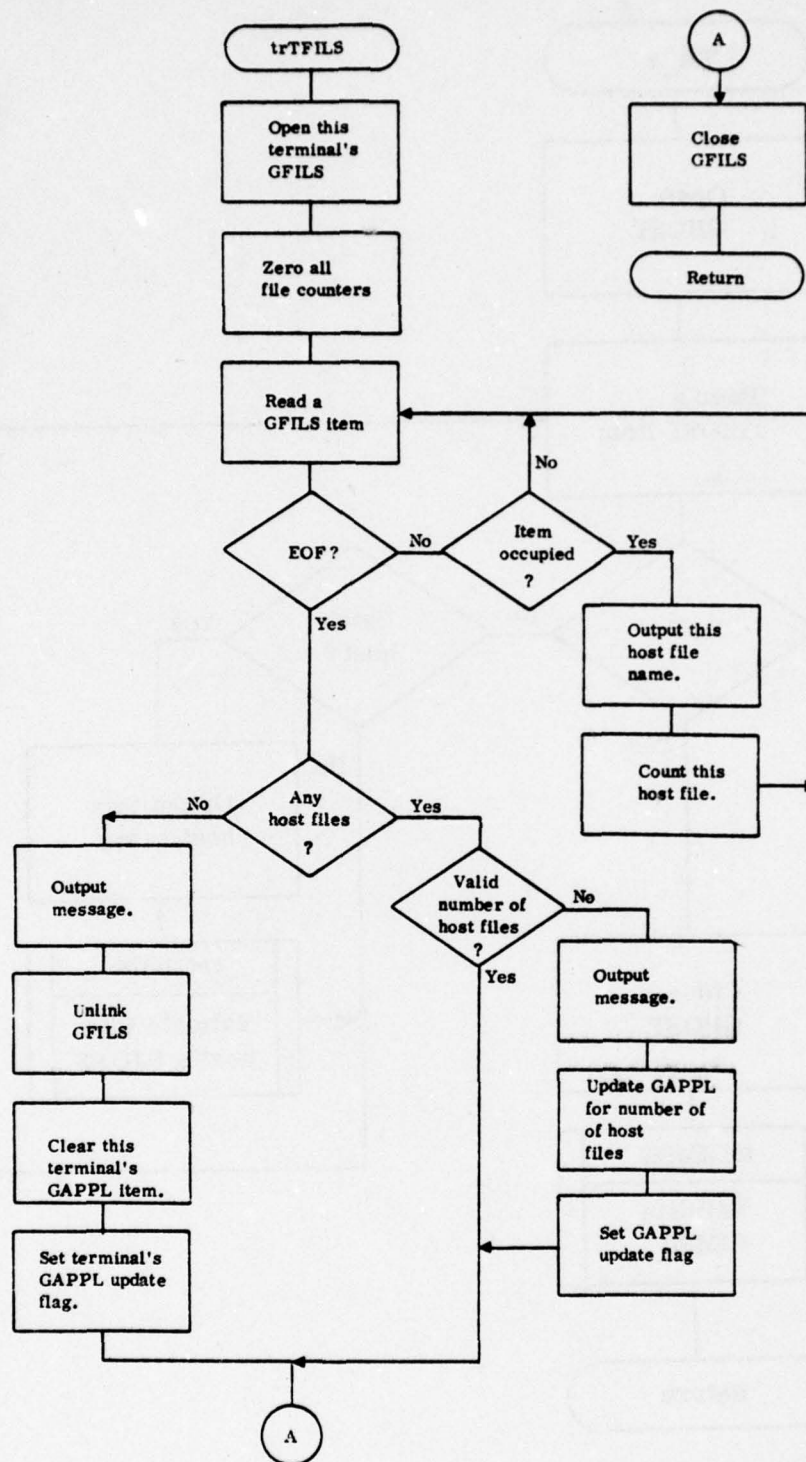


Figure 43. TRCC2 Process Data Flow (Sheet 5 of 16)

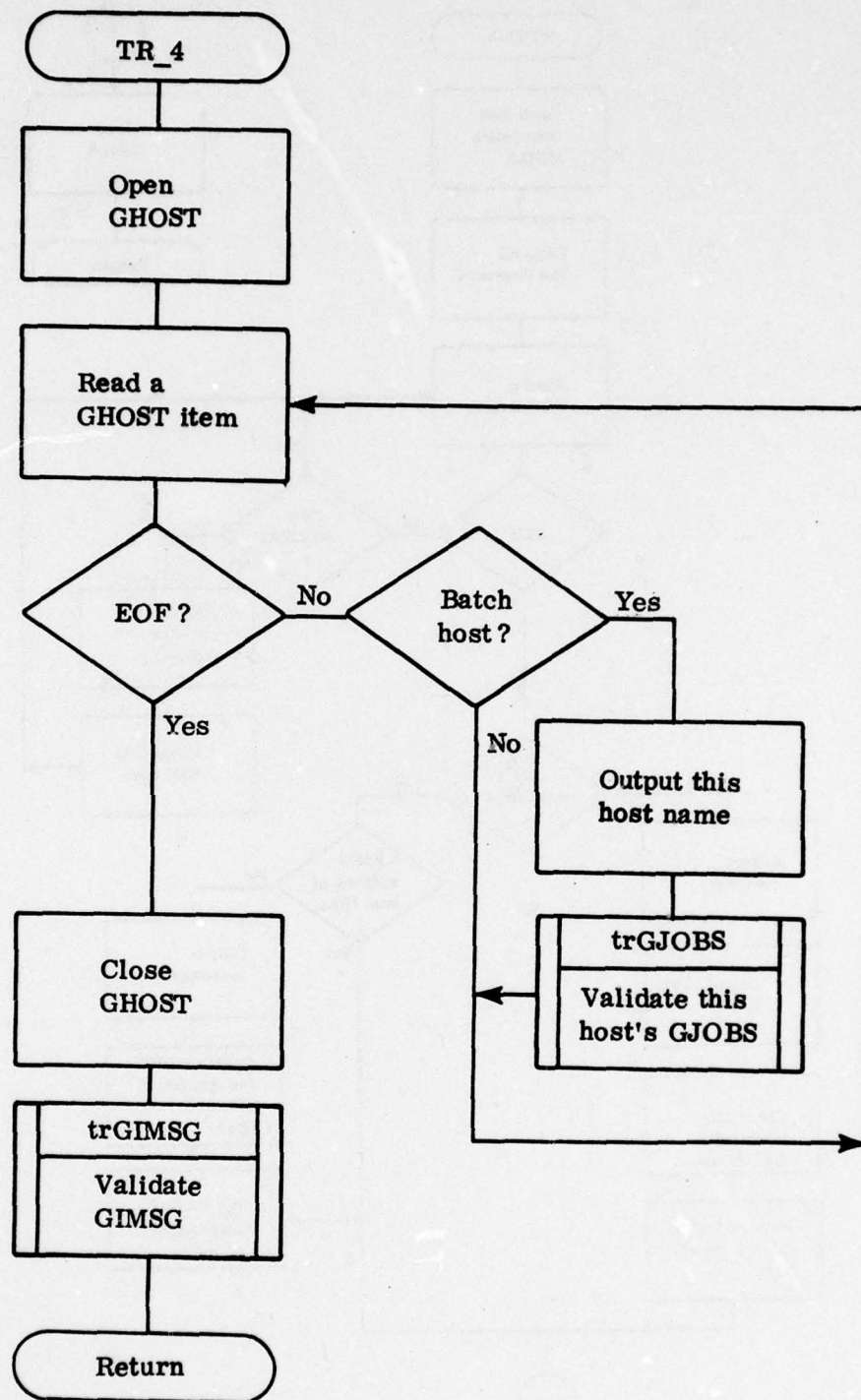


Figure 43. TRCC2 Process Data Flow (Sheet 6 of 16)

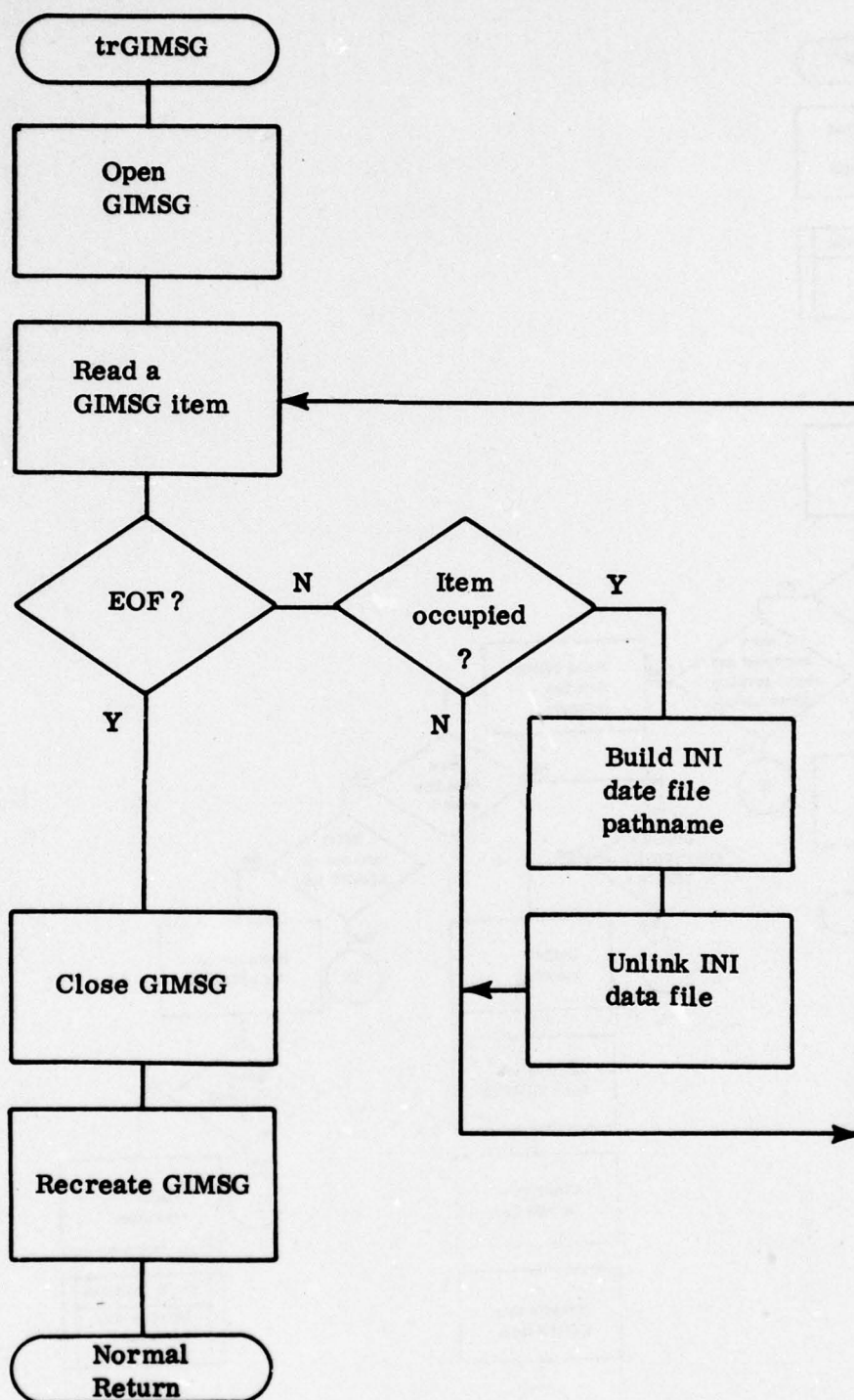


Figure 43. TRCC2 Process Data Flow (Sheet 7 of 16)

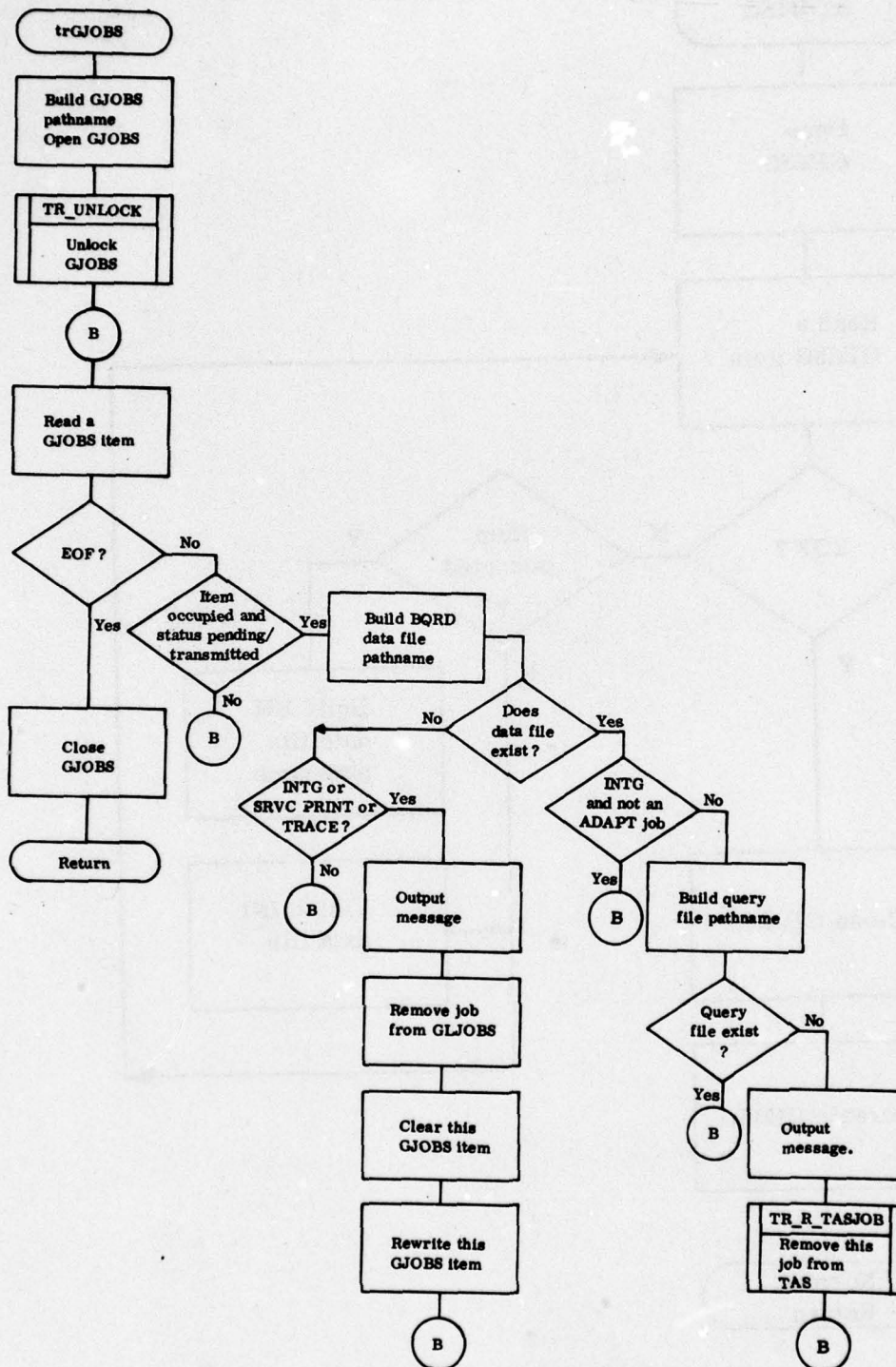


Figure 43. TRCC2 Process Data Flow (Sheet 8 of 16)

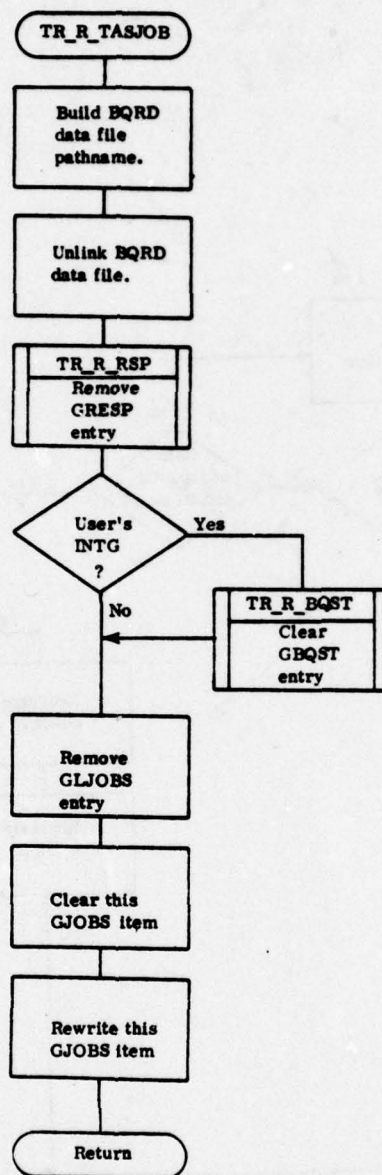


Figure 43. TRCC2 Process Data Flow (Sheet 9 of 16)

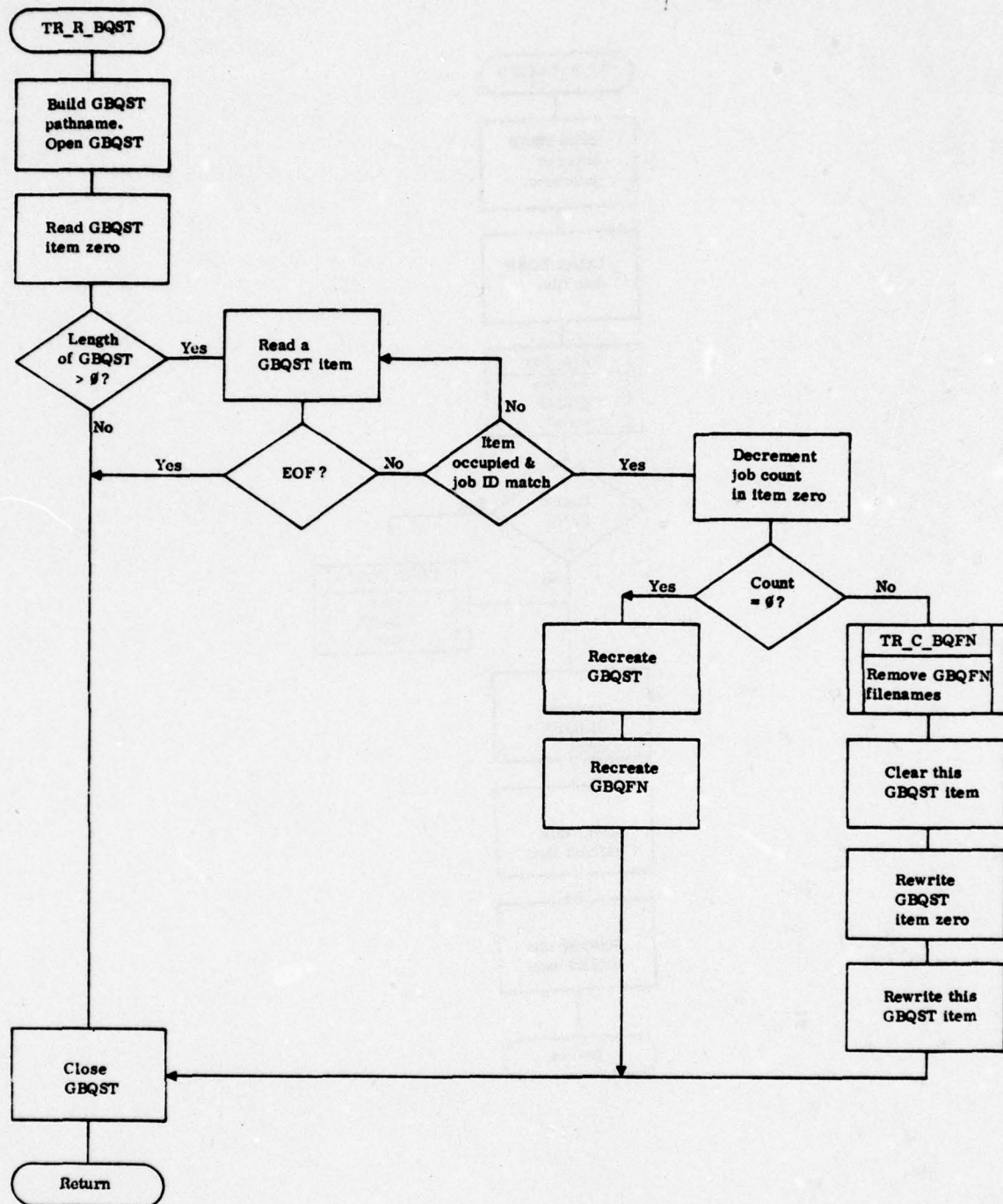


Figure 43. TRCC2 Process Data Flow (Sheet 10 of 16)

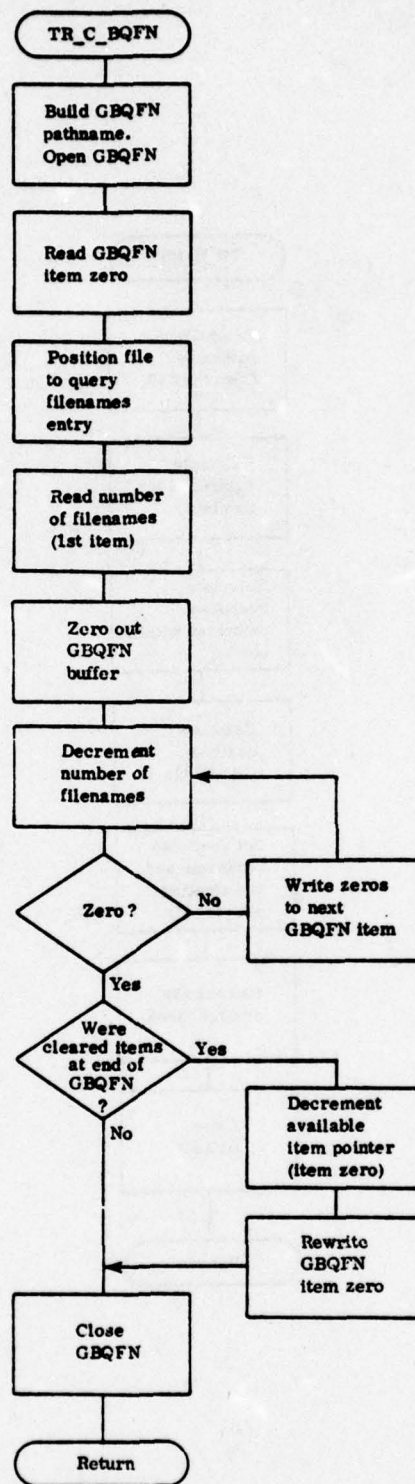


Figure 43. TRCC2 Process Data Flow (Sheet 11 of 16)

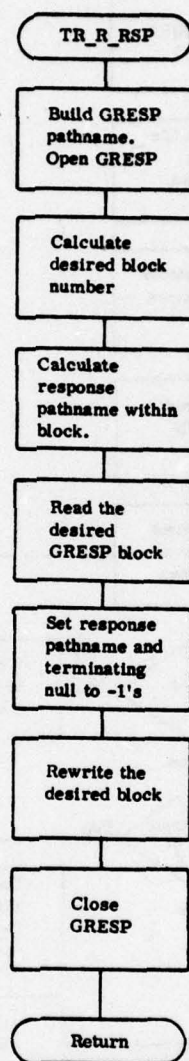


Figure 43. TRCC2 Process Data Flow (Sheet 12 of 16)

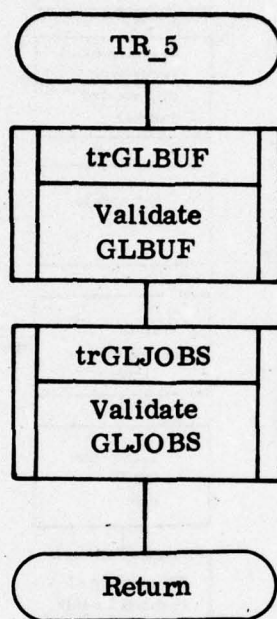


Figure 43. TRCC2 Process Data Flow (Sheet 13 of 16)

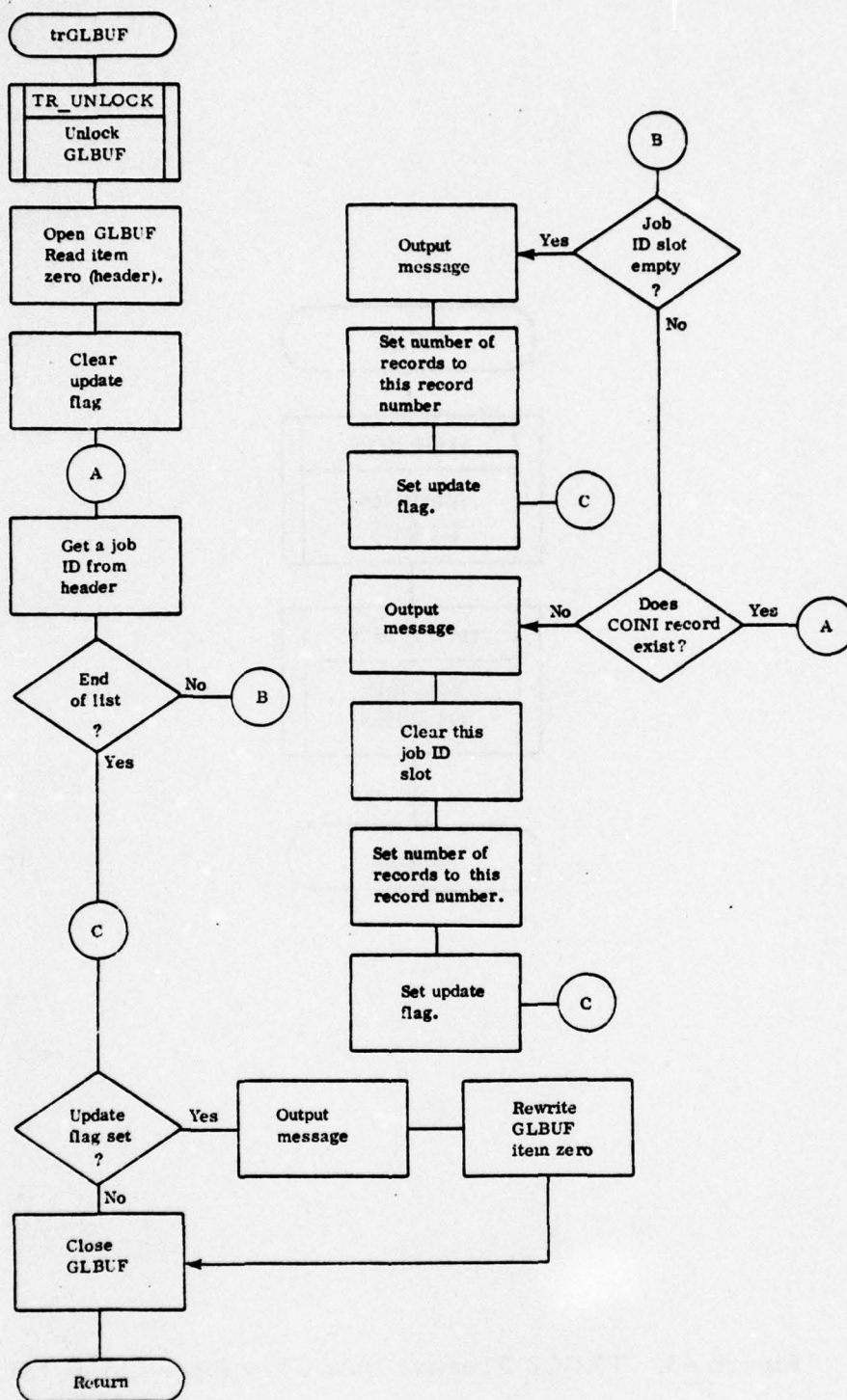


Figure 43. TRCC2 Process Data Flow (Sheet 14 of 16)

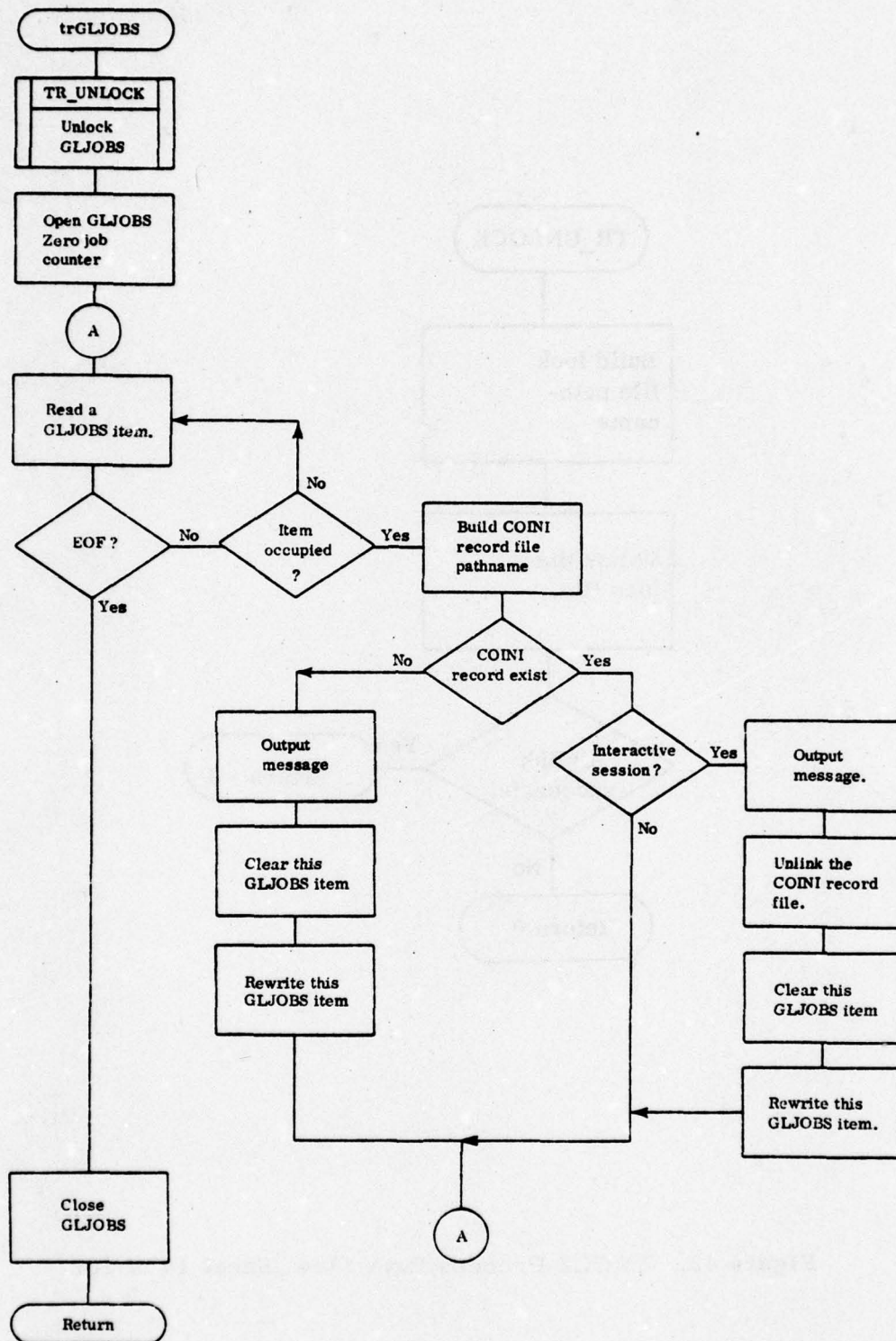


Figure 43. TRCC2 Process Data Flow (Sheet 15 of 16)

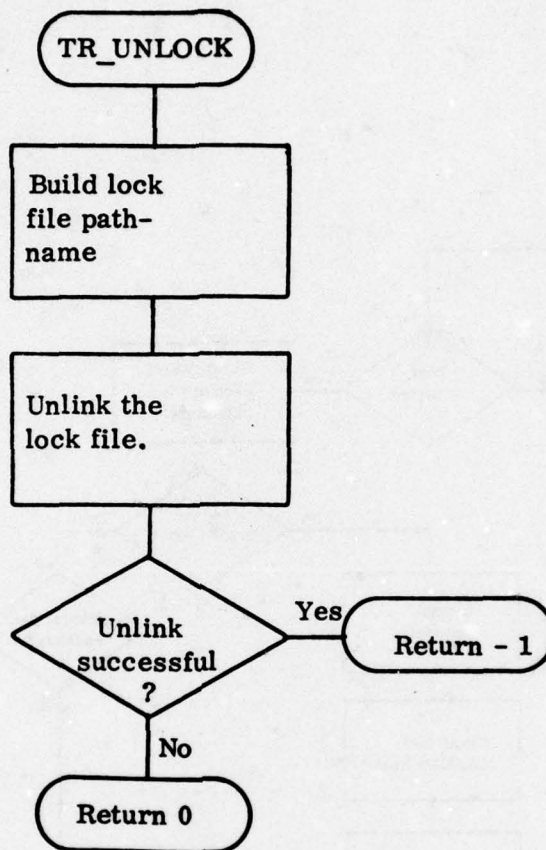


Figure 43. TRCC2 Process Data Flow (Sheet 16 of 16)

REMOVE (Remove Job/Text File)

The REMOVE process processes the REMOVE command which is invoked to delete a specific user's edit file or query/response files. The REMOVE command allows the TASMMASTER to selectively delete any files which are unnecessarily cluttering the TAS file system. If successful, the REMOVE process notifies both the affected user and the TASMMASTER of the deletion.

GLOBAL DATA USAGE - The REMOVE process uses the following global data structures.

- GBQST - user batch query statuses
- GBQFN - user batch query filenames
- GFILE - user text file descriptions
- GUIDES - user descriptions
- GMESG - user message descriptions

LOCAL DATA USAGE - The REMOVE process uses the character* array RMBUF to build the message buffer for the user whose file is deleted.

GENERAL PROCESS FLOW - The REMOVE process determines whether the command requester is the TASMMASTER; if not, a restricted-command diagnostic is output and REMOVE terminates. If the user is the TASMMASTER, REMOVE next insures that the TASMMASTER had input two parameters when requesting the REMOVE command. If any parameters are missing, the TASMMASTER is notified and REMOVE terminates. Having made the initial

validation, REMOVE opens GUES and then calls the global function GFLOCK to lock GUES. A search of GUES by REMOVE determines whether the first input parameter is a valid TAS sni. If the first argument is invalid, the appropriate diagnostic is output and the REMOVE process terminates.

If the user exists, a check is made to determine whether or not the user is logged off. If the user is logged off, the user status GLOGG is set to UPDATE and the user's GUES item is written out. GUES is unlocked by the global function GFUNLOCK.

The next phase of REMOVE locks GFILE and searches the user's GFILE to determine whether the second input parameter is a user text file name. If a match is found, the edit file is unlinked, and the GFILE entry is cleared and written. The edit file counter is decremented and the first GFILE item is written. In the special case where the counter is decremented to zero, GFILE is simply recreated. In any case, GFILE is unlocked via GFUNLOCK. A message, stored in RMBUF, is written to a file situated in the user's message directory and the user's GMESG is updated to include the new message entry. The REMOVE process informs the TASMMASTER of the deletion and then terminates.

If the second input argument is not found in GFILE, the function GFUNLOCK unlocks GFILE. In turn, file GBQST is locked and read. REMOVE attempts to match the second input parameter with a batch query JOBID. If a match is not found, a diagnostic is printed, notifying the TASMMASTER that the input parameter cannot be located among the specified user's edit or query/response files. If a match on the input argument and a query JOBID is made, REMOVE calls the BQRD process to cancel the query if the query's status is answered, aborted, or pending. BQRD need not be called if the query's status is delivered, lost, or logged out. REMOVE

unlinks the query and response files, clears and writes out the GBQST entry, and decrements the query file counter. If the counter is now zero, GBQST is essentially empty and GBQST and GBQFN are recreated. Otherwise, the counter is written out in the first GBQST entry. Also, the query's filenames are deleted from GBQFN. File GBQST is unlocked. As in the previously described situation, the affected user is notified of the deleted file via a message file. After informing the TASMASTER of the deleted query file, the REMOVE process terminates. See figure 44 for data flow.

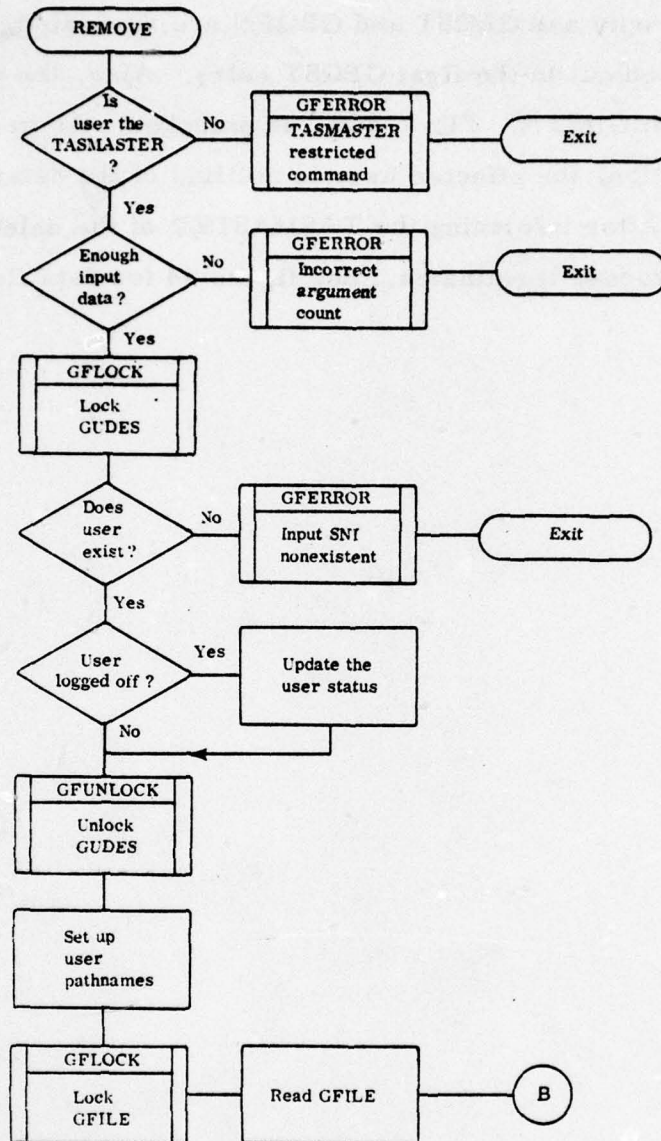


Figure 44. REMOVE Process Data Flow (Sheet 1 of 2)

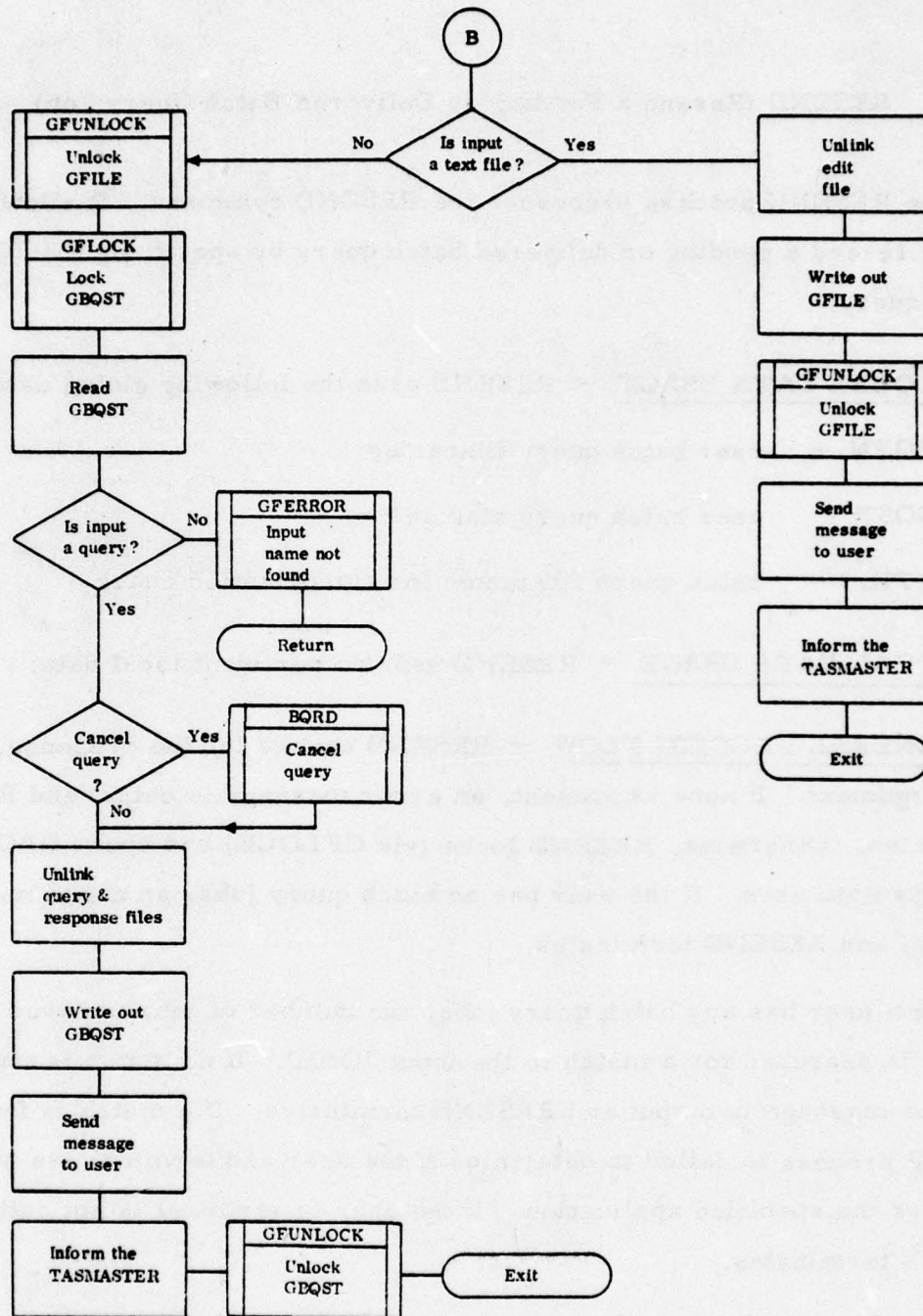


Figure 44. REMOVE Process Data Flow (Sheet 2 of 2)

RESEND (Resend a Pending or Delivered Batch Query Job)

The RESEND process processes the RESEND command. It allows a user to resend a pending or delivered batch query by specifying the JOBID of that query.

GLOBAL DATA USAGE - RESEND uses the following global data:

GBQFN - user batch query filenames

GBQST - user batch query statuses

GQFIL - batch query filenames for authorization check

LOCAL DATA USAGE - RESEND uses no pertinent local data.

GENERAL PROCESS FLOW - RESEND checks for the presence of an input argument. If none is present, an error message is output and RESEND terminates. Otherwise, RESEND locks (via GFLOCK) and opens GBQST and reads its item zero. If the user has no batch query jobs, an error message is output and RESEND terminates.

If the user has any batch query jobs, the number of jobs is saved and GBQST is searched for a match to the input JOBID. If no match is found, an error message is output and RESEND terminates. If a match is found, the AAP process is called to determine if the user and terminal are authorized to access the specified application. If the user or terminal is not authorized, RESEND terminates.

If both are authorized to access the application, GBQFN is opened and the first item for this job is read. If more filename items are present

in GBQFN, they too are read and GBQFN is closed. GQFIL is then created and the GBQFN filename items for this job are written into GQFIL. The AAP process is called to determine if the user and terminal are authorized to access the batch host files specified in the query (passed to AAP via GQFIL). If the user or terminal is not authorized access to the batch host files, GQFIL is unlinked and RESEND terminates.

If the user and terminal are authorized access to the specified batch host files, a test is made to determine if the job is pending. If it is not, and the job is not delivered, an error message is output and RESEND terminates. Otherwise, a "job has been purged" message is output.

If the job was pending, the BQRD process is called to cancel the job. A "job has been cancelled" message is then output.

If the job was either pending or delivered, GFJOBID is called to obtain a new TAS JOBID and BQRD is called with the new INTG. Then a GBQST item for the new job is formatted and written out, a link is made to the new query file and the old query file is unlinked.

Finally, GQFIL is unlinked and RESEND terminates. See figure 45 for data flow.

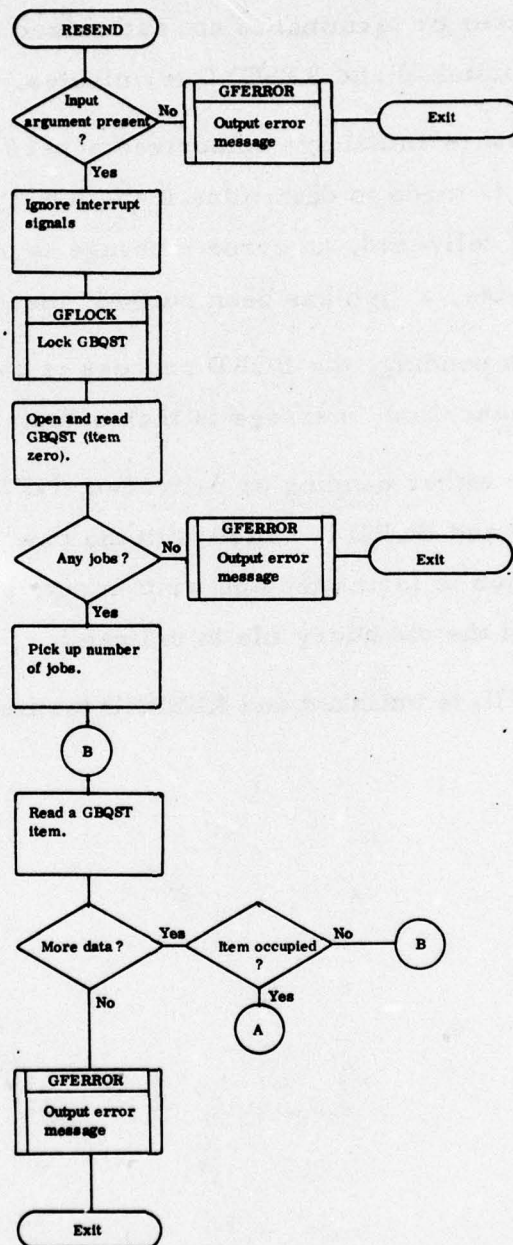


Figure 45. RESEND Process Data Flow (Sheet 1 of 2)

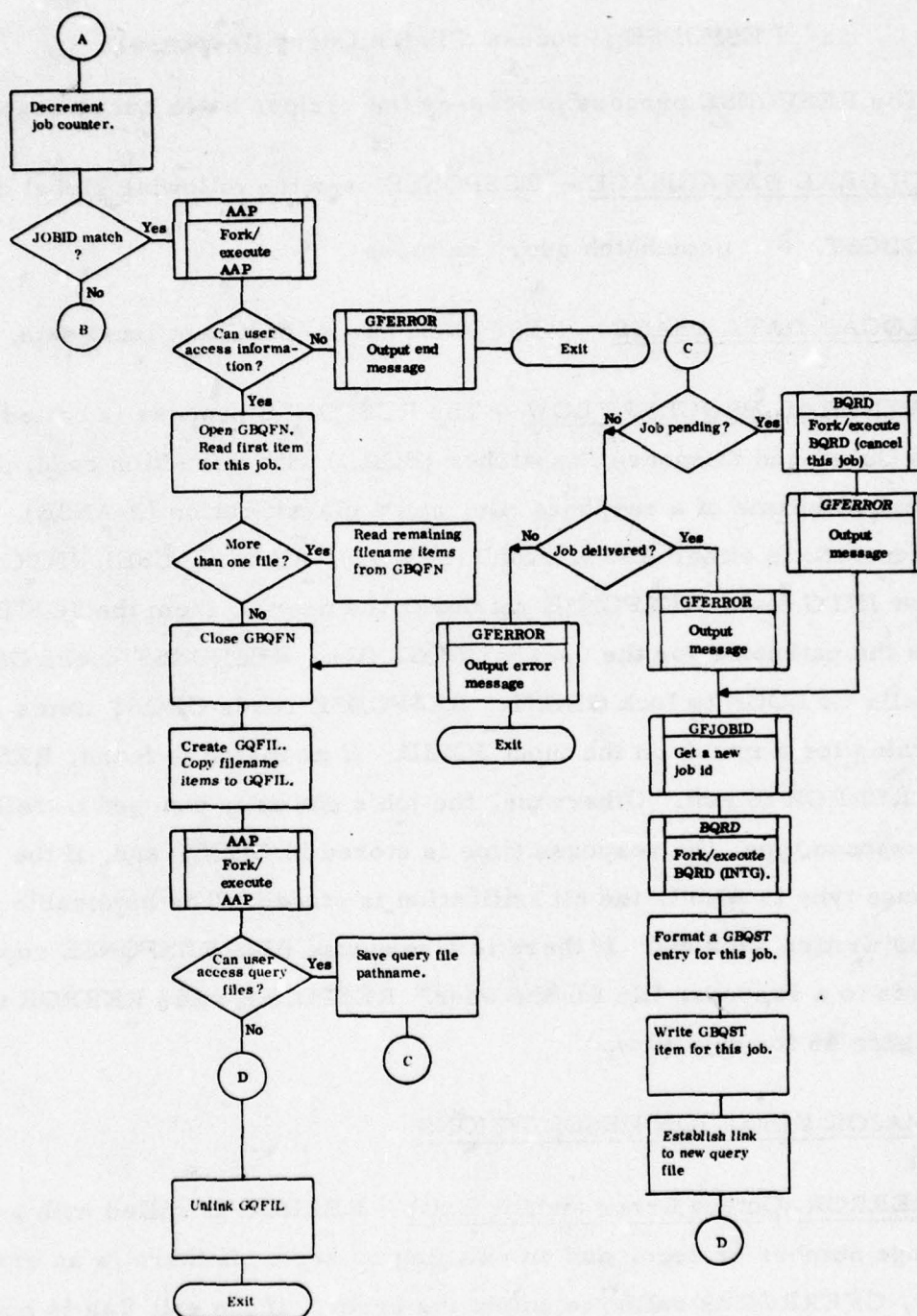


Figure 45. RESEND Process Data Flow (Sheet 2 of 2)

RESPONSE (Process a Batch Query Response)

The RESPONSE process processes the various batch query responses.

GLOBAL DATA USAGE - RESPONSE uses the following global data:

GBQST - user batch query statuses

LOCAL DATA USAGE - RESPONSE has no pertinent local data.

GENERAL PROCESS FLOW - The RESPONSE process is called by the Batch Query and Response Dispatcher (BQRD) with a function code, JOBID, optional pathname of a response file, and a classification (if ANSR). The function code is either ANSR, ABRT (FAULT), ABRT (CAUSE), INTG logged out, or INTG lost. RESPONSE retrieves the user ID from the JOBID and builds the pathname for the user's GBQST file. RESPONSE opens GBQST and calls GFLOCK to lock GBQST. RESPONSE reads GBQST items searching for a match on the input JOBID. If no match is found, RESPONSE calls RERROR to exit. Otherwise, the job's status is changed to reflect the response type, the response time is stored in GBQST and, if the response type is ANSR, the classification is stored. The applicable GBQST item is written back out. If there is a response file, RESPONSE copies its contents to a response file for the user. RESPONSE calls RERROR to exit. See figure 46 for data flow.

MAJOR FUNCTION DESCRIPTIONS

RERROR (Output Error and/or Exit) - RERROR is called with a message number or zero, and an exit flag or zero. If there is an error to output, GFERROR is called to output the error. If the exit flag is not set, RERROR returns. Otherwise, if GBQST is locked, GFUNLOCK is called to unlock it, and RERROR exits.

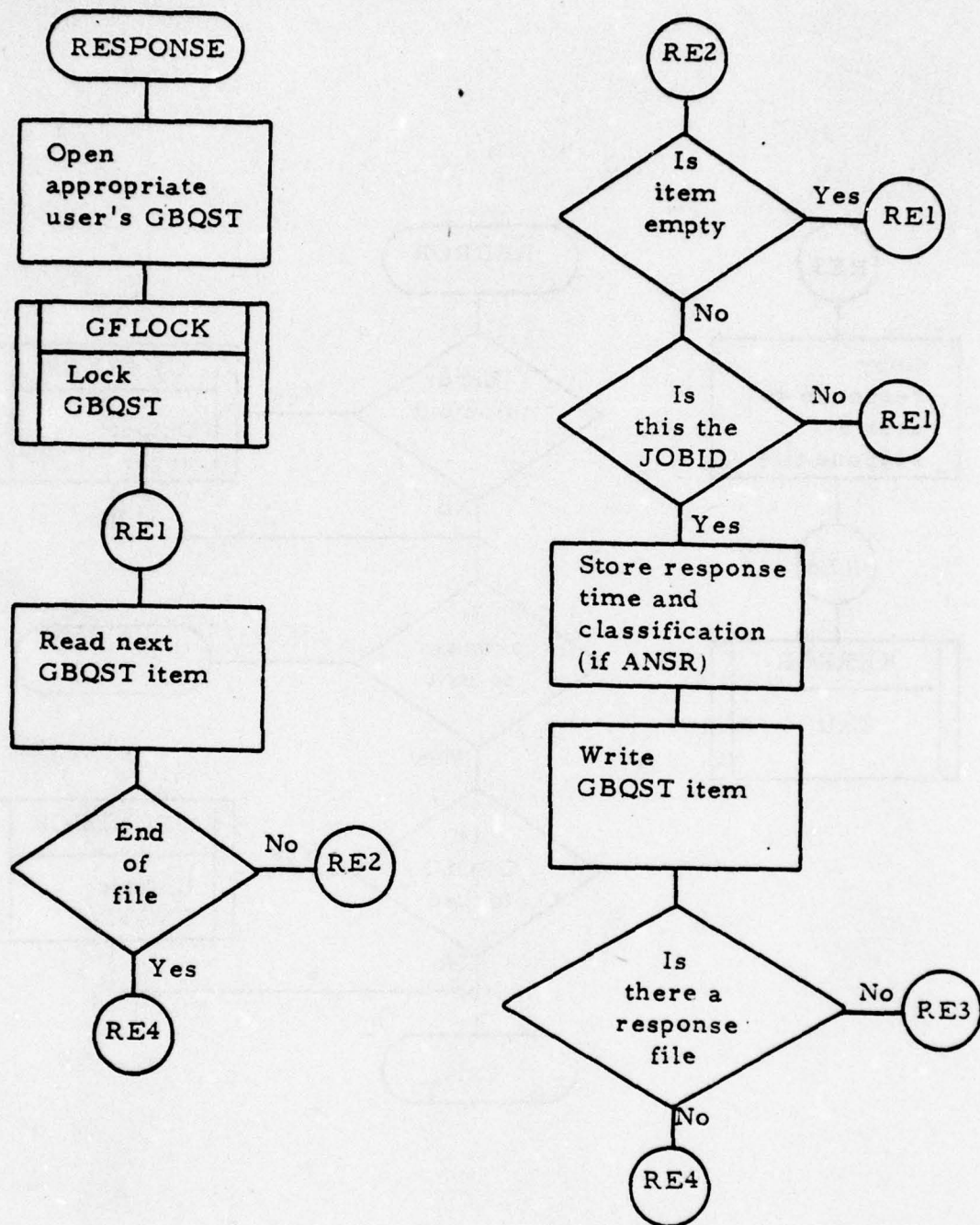


Figure 46. RESPONSE Process Data Flow (Sheet 1 of 2)

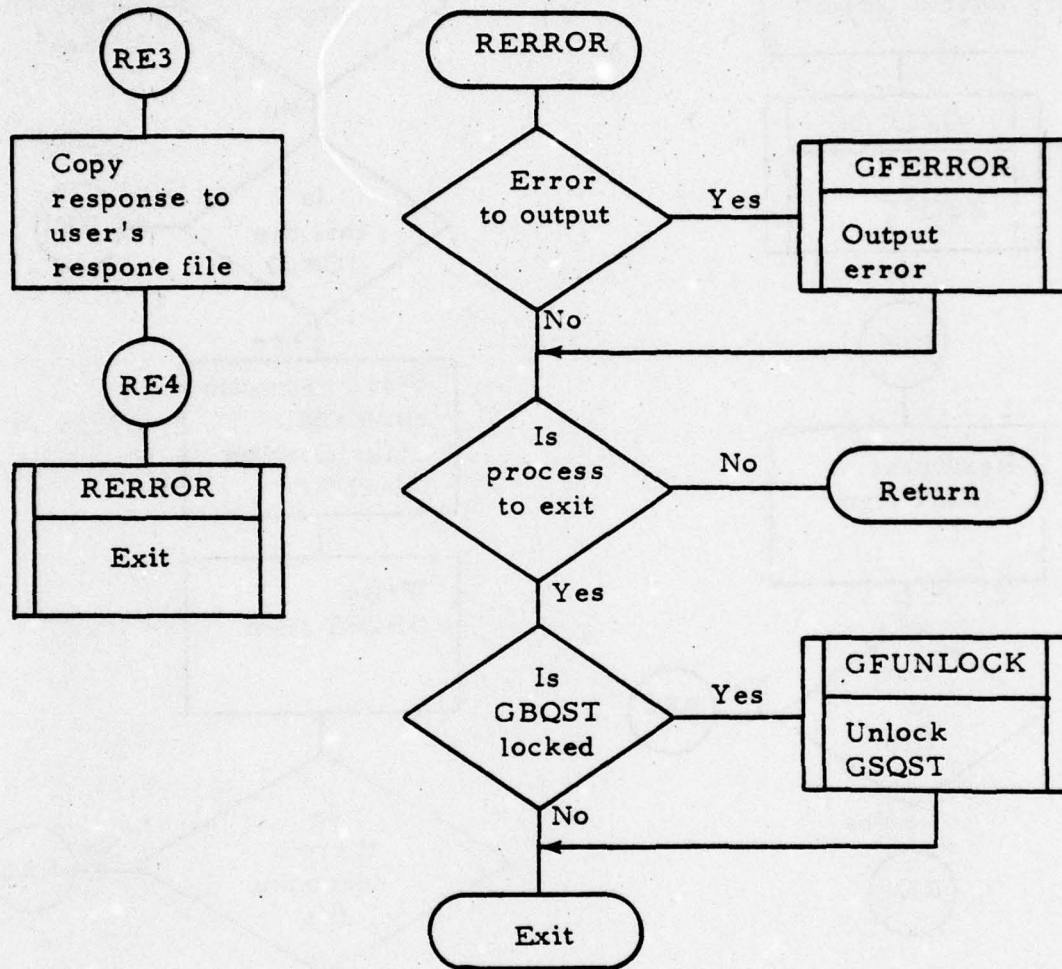


Figure 46. RESPONSE Process Data Flow (Sheet 2 of 2)

SAVE (Save the Contents of a Batch Query in a Text File)

The SAVE process processes the SAVE command. It allows a user to save the contents of one of his batch queries in a specified text file.

GLOBAL DATA USAGE - SAVE uses the following global data:

GBQST - user batch query statuses

GFILE - user text file descriptions

LOCAL DATA USAGE - SAVE uses no pertinent local data.

GENERAL PROCESS FLOW - SAVE tests for the presence of two input arguments. If there are not two input arguments, SAVE outputs an error message and terminates. The input filename is validated to ensure that it is composed of only letters, digits, and periods (.), and is not over 13 characters long. If the filename is invalid, an error message is output and SAVE terminates.

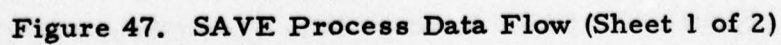
If the save filename is valid, GBQST is locked and opened, and its item zero is read. If there are no jobs for this user, an error message is output and SAVE terminates. Otherwise, the number of jobs is saved and GBQST is searched for a match to the input JOBID. If a match is not found, an error message is output and SAVE terminates.

If a match to the input JOBID is found, GFILE is locked, opened and its item zero is read. If the user has no text files, the number of files is set to zero; otherwise, if he has too many files, an error message is output and SAVE terminates.

The file count is then saved and the user's GFILE is searched for a match on the save filename or for the last unoccupied item. If a match is found for the save filename, the user is given the option of overwriting

the file. If he does not wish to do so, a message is output and SAVE terminates: otherwise, the old file is unlinked, the query file for the specified job is copied to the specified text file, the user is notified that the query has been saved and SAVE terminates.

If a match to the input filename is not found in GFILE, the index of the last available GFILE item is saved, a new GFILE item for the file is created, the number of user text files is incremented in GFILE item zero and item zero is rewritten. The contents of the query file for the specified job are copied to the new user text file, the user is informed that the batch query job has been saved and SAVE terminates. See figure 47 for data flow.



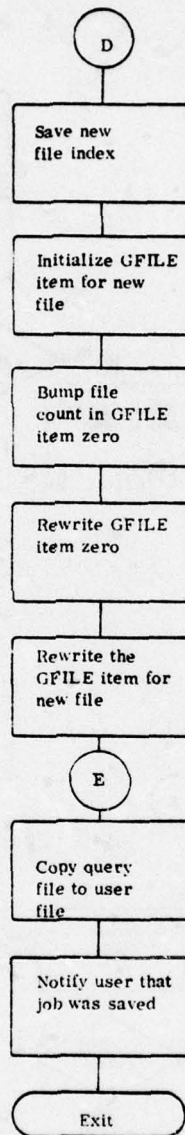


Figure 47. SAVE Process Data Flow (Sheet 2 of 2)

SHOW (Display a Text File or Batch Query)

The SHOW process processes the SHOW command. It displays the contents of a specified query or of a specified text file. The TASMMASTER also has the option to specify a user in addition to a query or text file.

GLOBAL DATA USAGE - SHOW uses the following global data:

GBQST - user batch query statuses
GFILE - user text file descriptions
GUIDES - user descriptions

LOCAL DATA USAGE - SHOW has no pertinent local data.

GENERAL PROCESS FLOW - SHOW first determines whether any input parameters exist. If they do not exist, a diagnostic is output and the process terminates. If two input parameters exist, the command user must be the TASMMASTER. If not a diagnostic is output and the process then terminates. If the user is the TASMMASTER, the given sni is located in GUIDES and the user pathnames are established. If the input sni is not located, an error message is output and the process terminates.

Whether the command is input by the TASMMASTER to view another user's file or by a user to view one of his own files, the remaining program logic is identical. SHOW reads the user's GBQST file and searches for the specified JOBID. If a JOBID match is made, then SHOW reads the query file associated with the JOBID, outputs the contents of the file and terminates. If the input parameter is not a JOBID, SHOW reads the user's GFILE file and searches to see if the input parameter is a filename. If not a diagnostic is output and SHOW terminates. Otherwise, SHOW reads the specified user file, outputs the contents of the file and terminates. See figure 48 for data flow.

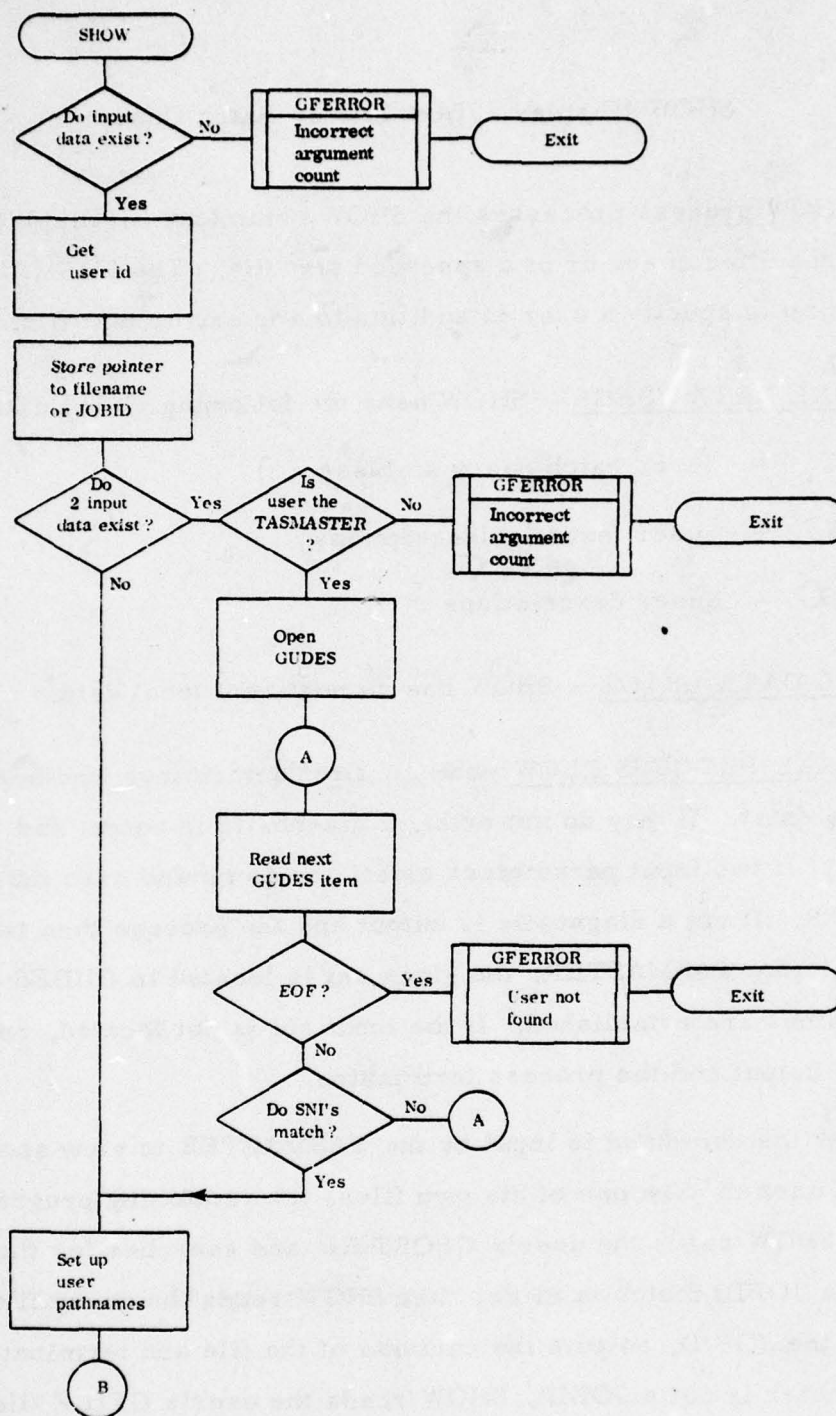


Figure 48. SHOW Process Data Flow (Sheet 1 of 2)

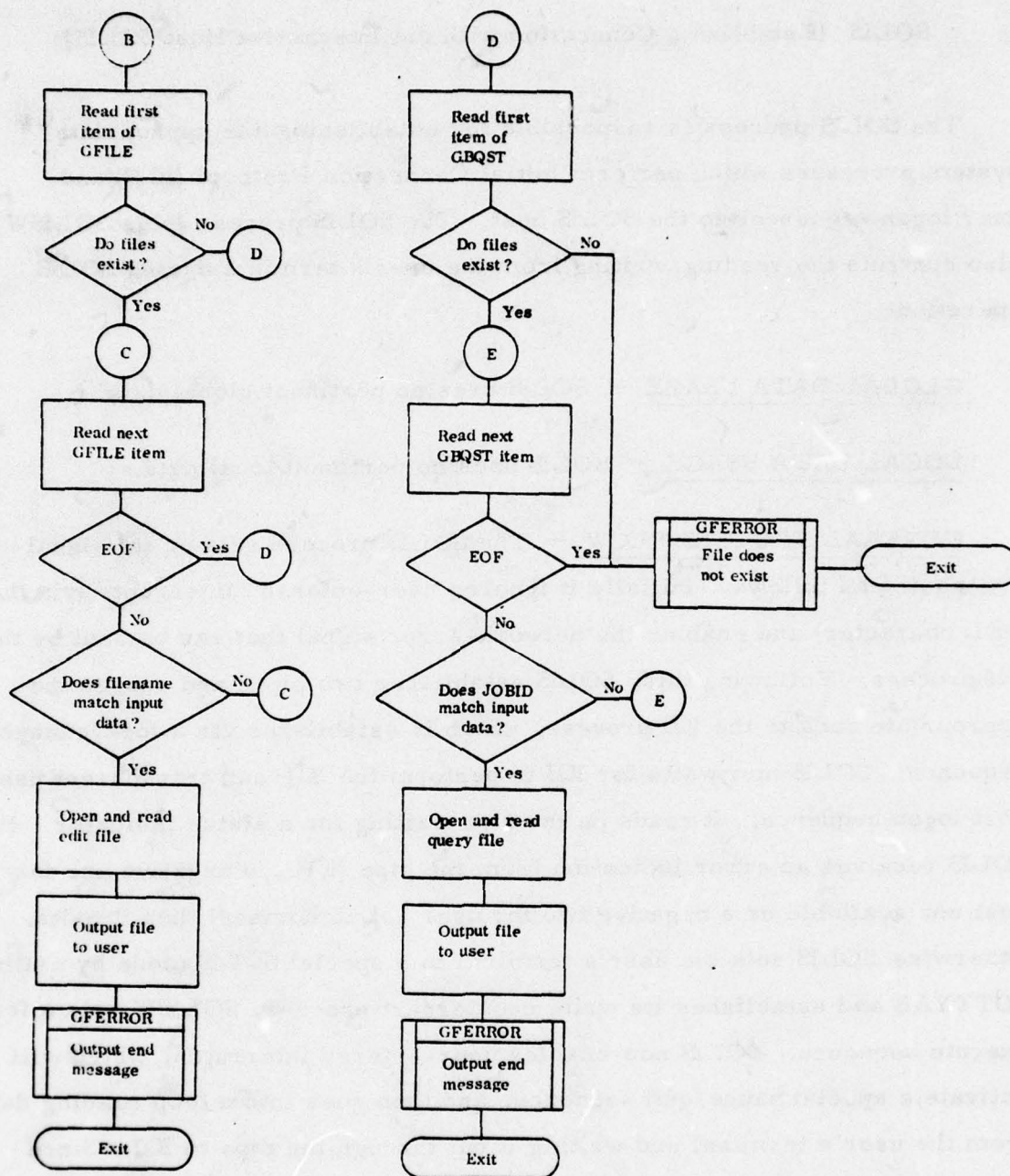


Figure 48. SHOW Process Data Flow (Sheet 2 of 2)

SOLIS (Establish a Connection with the Interactive Host SOLIS)

The SOLIS process is responsible for establishing the appropriate system processes which perform Initial Connection Protocol (ICP) and user logon sequences to the SOLIS host. The SOLIS process (plus SOLISW) also controls the reading/writing from the user's terminal during SOLIS operations.

GLOBAL DATA USAGE - SOLIS uses no pertinent global data.

LOCAL DATA USAGE - SOLIS uses no pertinent local data.

GENERAL PROCESS FLOW - The SOLIS process sets up its signal processing as follows. Initially it ignores user-entered interrupts (via the DEL character) and enables the network-error signal that can be sent by the IQI process. Following this, SOLIS establishes two pipes and passes the appropriate ends to the IQI process, which it establishes via a fork/execute sequence. SOLIS now waits for IQI to perform the ICP and transparent user host logon sequence. It reads on the pipe waiting for a status indicator. If SOLIS receives an error indication from the pipe (i. e., a negative one for host not available or a negative two for user not authorized) then it exits. Otherwise SOLIS sets the user's terminal to a special SOLIS mode by calling SOTTYAB and establishes its write counterpart process, SOLISW, via a fork/execute sequence. SOLIS now enables user-entered interrupts, which will activate a special pause/quit sequence, and then goes into a loop reading data from the user's terminal and writing it out through the pipe to IQI. Since the user's terminal is now in a raw mode (refer to function SOTTYAB), SOLIS must scan each character read in from the terminal for a DEL character.

If a DEL character is encountered and is one of the first three characters read in from the screen, then SOLIS activates the DEL signal processor. This function, SOINTRPT, performs the pause/quit sequence with the user. This loop continues until the user enters a DEL character to quit (or to pause), the user enters a period-command ('. <ETX>') to the SOLIS system which in turn closes the connection, or either IQI or IQIR encounters a network error. The latter two are detected by IQI or IQIR which in turn notifies SOLIS via a signal. See figure 49 for data flow.

MAJOR FUNCTION DESCRIPTIONS

SOINTRPT (Process the Interrupt Signal from the Terminal) - This function is responsible for processing user-entered DEL characters. SOINTRPT can either be called from a true signal (when the terminal is in a full duplex/echo non-raw mode) or it can be called internally from the main when the DEL character is detected while in the special SOLIS terminal raw-mode. Upon entry of this function the SOLISW process is signalled to stop writing to the user's screen and the terminal is placed back into a normal TAS operating mode by calling SOTTYNM. At this point the user is prompted with a 'pause or quit' question and SOINTRPT looks for the proper response. If the user enters a 'q' for quit then the termination cycle is entered by a call to the general error/exit function SOERROR. If the user does not enter a 'q' then the pause mode is entered. SOINTRPT then invokes the TASHELL with a parameter of 'p' to indicate a pause state. From this point on SOINTRPT waits until the TASHELL exits from the pause state, hence completely blocking SOLIS processing. When the user resumes his SOLIS operation, the TASHELL exits and the SOLIS process continues

execution from the wait call in SOINTRPT. Now SOINTRPT checks to insure that TAS is not in a 'TAS down sequence' (where it exits if it is) and then goes into a 'continue or quit' sequence with the user. If the user enters a 'q', then SOINTRPT initiates the termination sequence by calling SOERROR. If not, then a signal is sent to SOLISW to reinitiate the screen writing sequence and the user's terminal is placed back into the special SOLIS mode via SOTTYAB. Finally, a 'R<ETX>' sequence is written out to the pipe to cause the retransmission of the last SOLIS screen.

SOERROR (Generalized Error and Exit Processor) - This function is responsible for processing error messages and terminating SOLIS. SOERROR first checks to see if the user is not in the pause state and the terminal is in the special SOLIS mode. For this situation the terminal is returned to the normal TAS operating mode via SOTTYNM. Next SOERROR determines if this is an error condition and outputs an error via GFERROR if applicable. Finally SOERROR determines if this is an exit condition. If it isn't, then the function returns to the calling function; otherwise it signals both the IQI and SOLISW processes to terminate (if they are present) and then waits until all processes have terminated. Then SOERROR checks to see if the user is in a pause mode. If the user is, SOERROR outputs 'Host has terminated your connection' to the user and then exits. Otherwise, it just exits.

SOTTYAB (Special SOLIS Terminal Mode Processor) - This function changes the user's terminal mode (via gtty/stty sequences) to a special SOLIS mode as follows:

1. activate 'raw mode'

2. inhibit 'echo mode'
3. inhibit 'NL mapping to CR-LF'

This function also outputs a message to the user informing him to place his terminal in a local mode. SOTTYAB saves the original terminal mode for SOTTYNM.

SOTTYNM (Normal TAS Terminal Mode Processor) - This function restores a terminal to the normal TAS operating mode. It uses the terminal mode saved by SOTTYAB. Following terminal mode restoration, this function clears the terminal of all potential formatted fields by issuing an escape-R sequence.

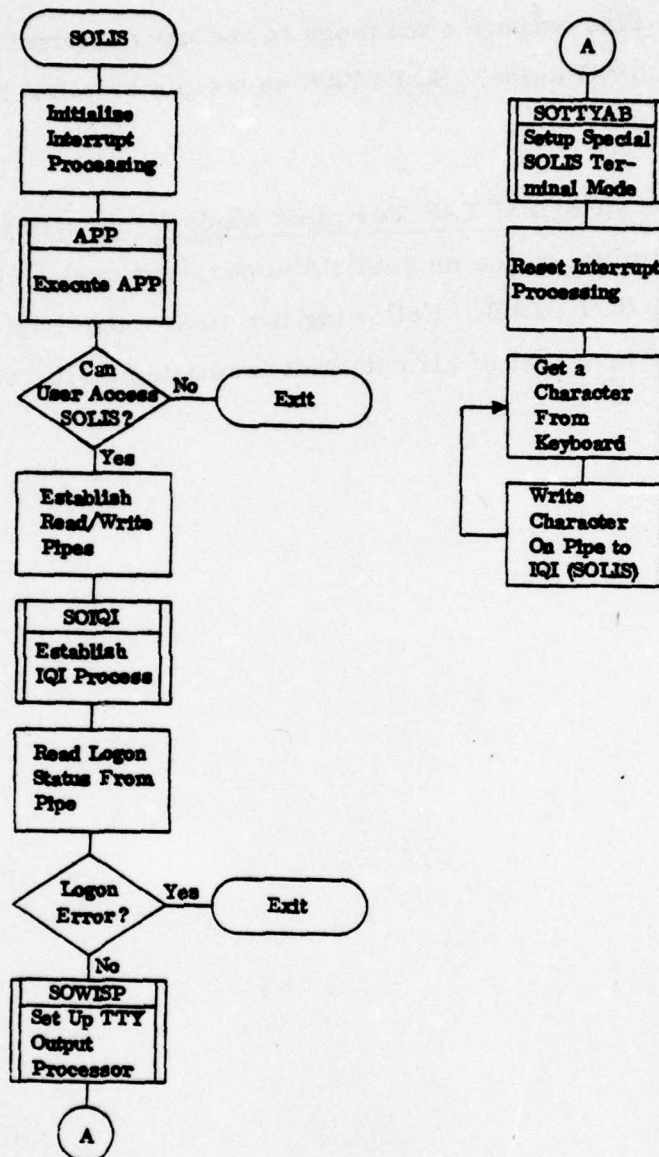


Figure 49. SOLIS Process Data Flow (Sheet 1 of 5)

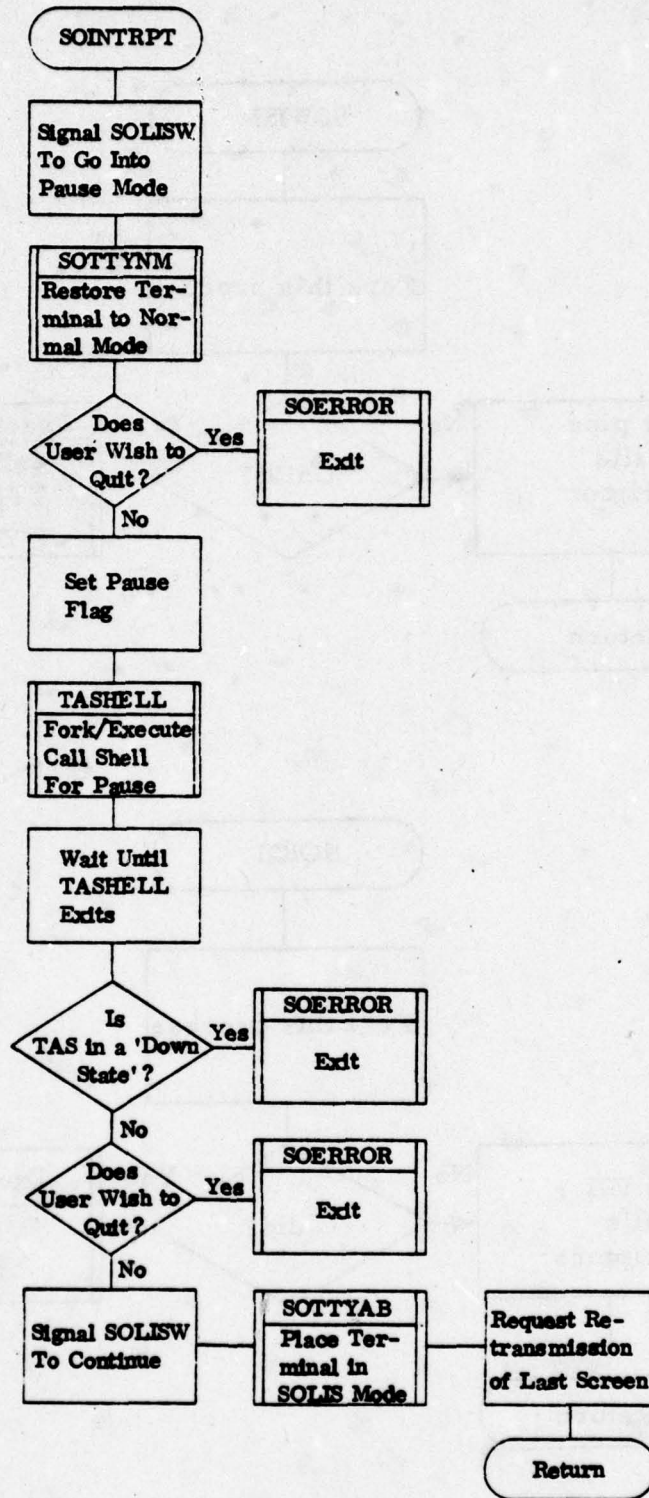


Figure 49. SOLIS Process Data Flow (Sheet 2 of 5)

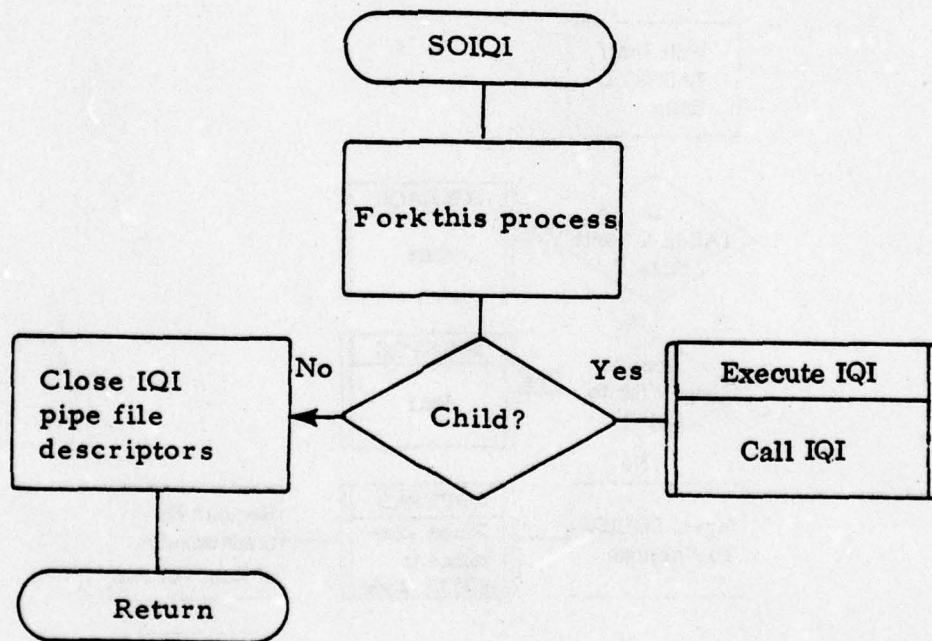
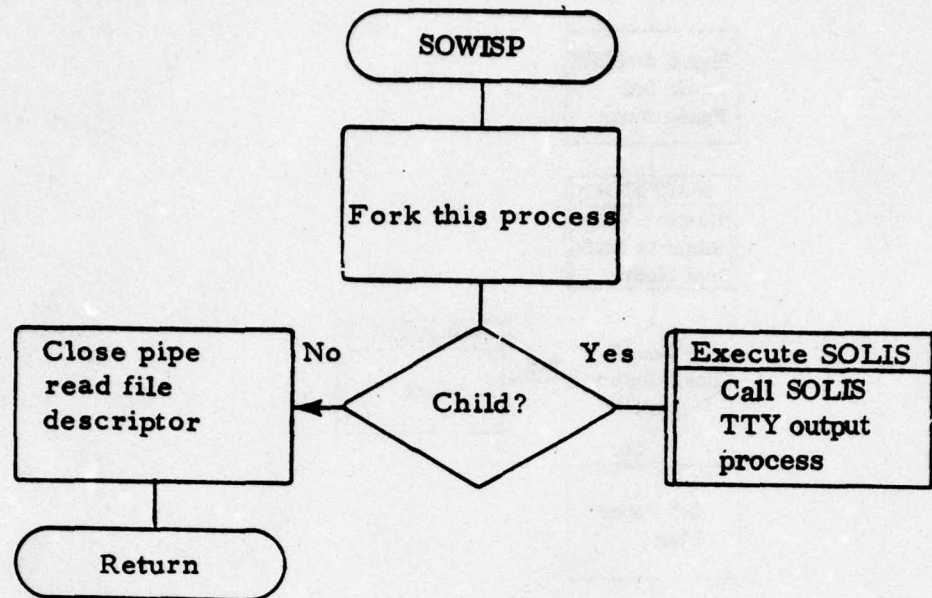


Figure 49. SOLIS Process Data Flow (Sheet 3 of 5)

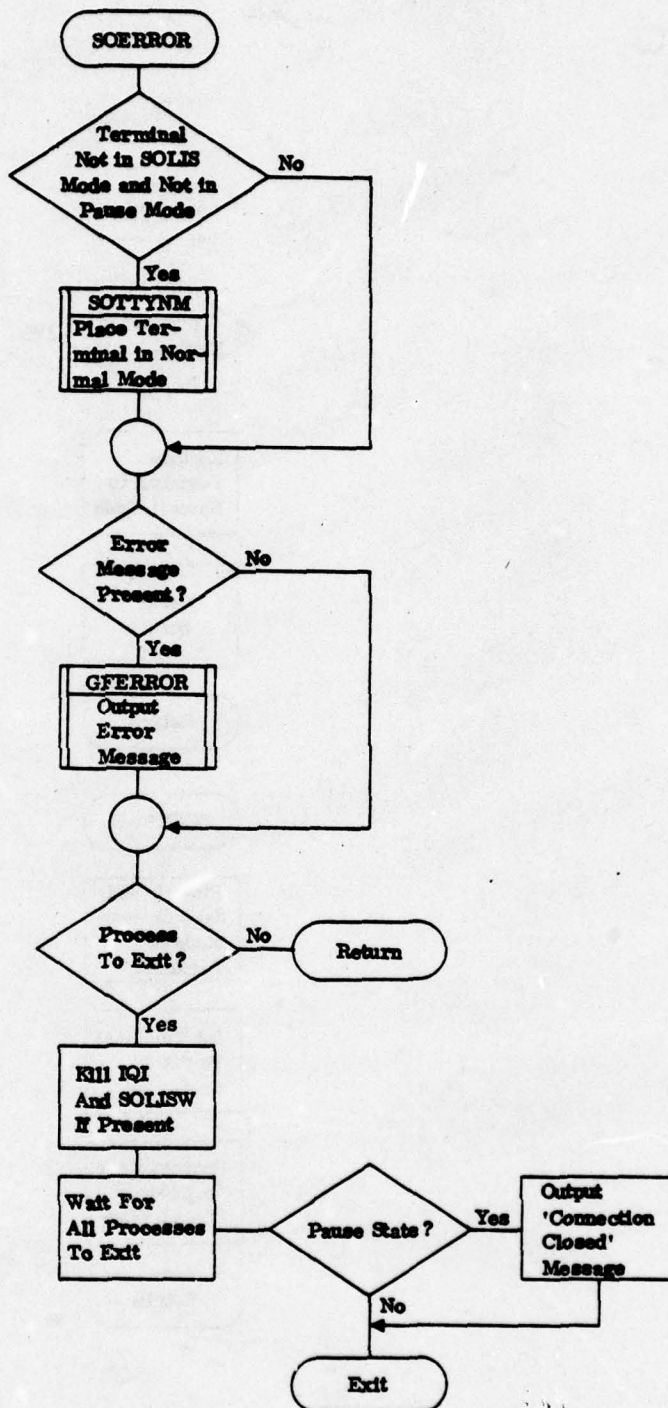


Figure 49. SOLIS Process Data Flow (Sheet 4 of 5)

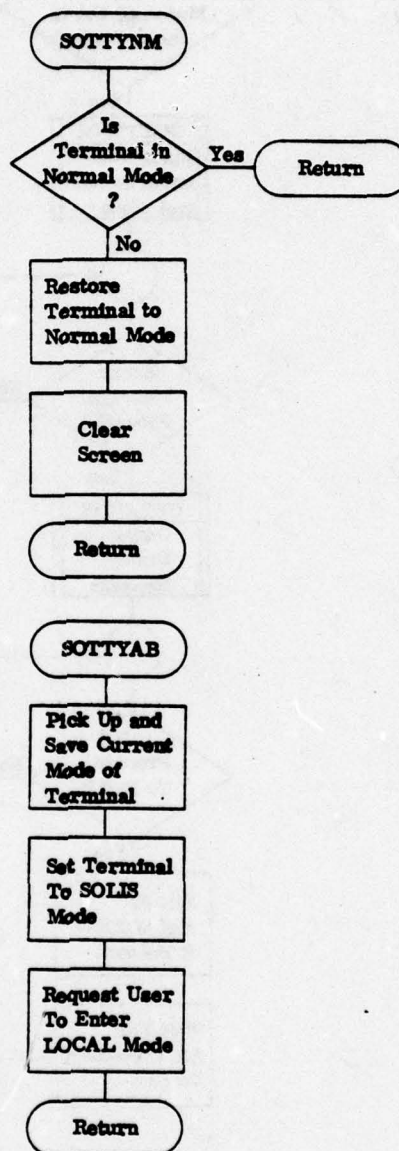


Figure 49. SOLIS Process Data Flow (Sheet 5 of 5)

SOLISW (Perform Terminal Writing for SOLIS Operations)

The SOLISW process is a system process which is responsible for writing network responses from the SOLIS host to the user's terminal.

GLOBAL DATA USAGE - SOLISW uses no pertinent global data.

LOCAL DATA USAGE - SOLISW uses no pertinent local data.

GENERAL PROCESS FLOW - SOLISW establishes its signal processing as follows. User interrupts via the DEL character are ignored. Both a terminate and pause signal are invoked and will be trapped by functions SOEMTINS and SOPAUSE, respectively. These two signals are generated by the SOLIS process which called SOLISW. The basic loop for SOLISW is a sequence where it reads data from a pipe and writes it to the user's terminal. Data from the pipe is written by IQIR and contains network data from the SOLIS host. This loop continues until one of two events occur. First, a terminate signal from SOLIS causes SOLISW to exit. This is caused either when the user desires to terminate the SOLIS session or if either IQI or IQIR encounters network errors. Second, a signal indicating that the user has gone into a pause state will alter this loop. For this situation SOLISW sets the pause flag which is tested in the pipe-read/terminal-write sequence. If it is set then the data read from the pipe is ignored. This continues until the user decides to resume his interactive session, which is indicated to SOLISW by the same pause signal. Then SOLISW continues with the original loop sequence of pipe-reading/terminal-writing. See figure 50 for data flow.

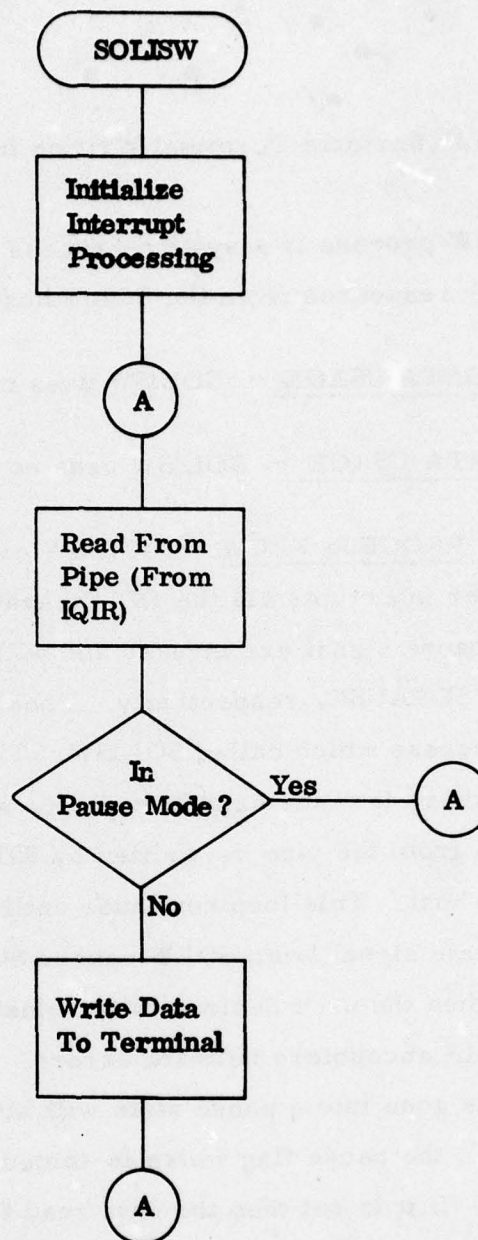


Figure 50. SOLISW Process Data Flow (Sheet 1 of 2)

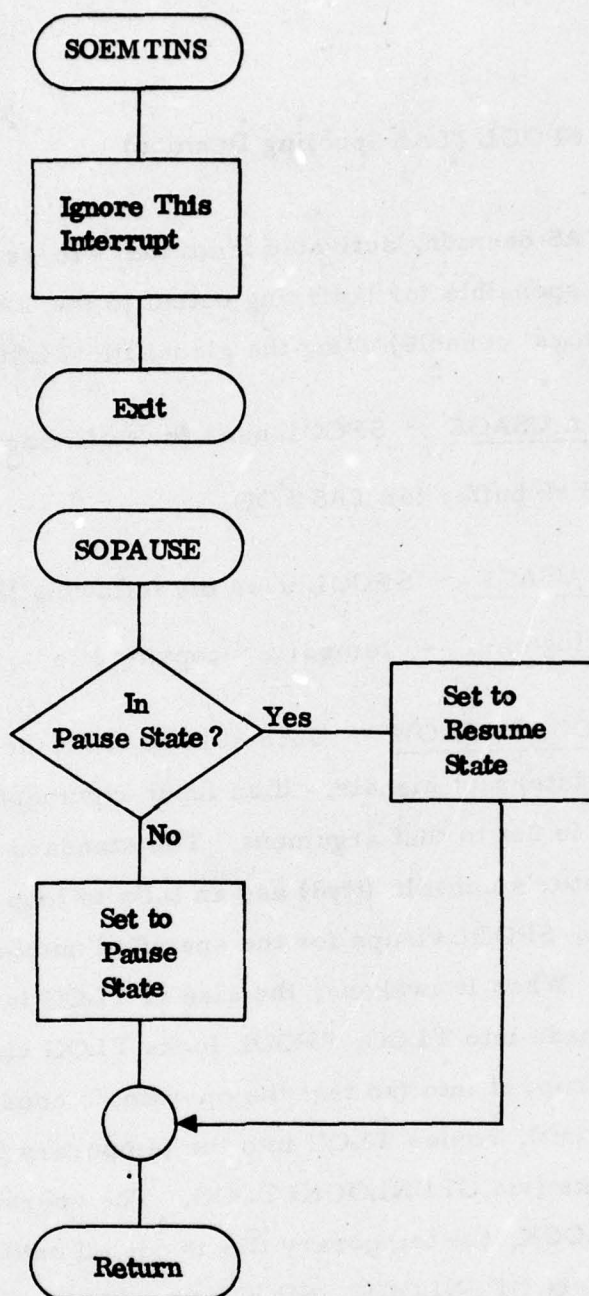


Figure 50. SOLISW Process Data Flow (Sheet 2 of 2)

SPOOL (TAS Spooling Daemon)

SPOOL is a TAS daemon, activated from the "etc/rc" UNIX initialization file, and is responsible for buffering output to the TAS LOG (which is actually the operators' console) using the global file TLOG.

GLOBAL DATA USAGE - SPOOL uses the following global data:

TLOG - disk buffer for TAS LOG

LOCAL DATA USAGE - SPOOL uses the following local data:

/usr/tsy/log/tlogtemp - temporary copying file

GENERAL PROCESS FLOW - Once started from the "/etc/rc" file, SPOOL ignores all interrupt signals. If an input argument has been specified, the spooling period is set to that argument. The standard I/O file descriptors are set to the operator's console (tty8) and an infinite loop is entered. At the start of the loop, SPOOL sleeps for the specified number of seconds (ten is the default). When it awakens, the size of TLOG is obtained. If entries have been made into TLOG, SPOOL locks TLOG via GFLOCK, creates a temporary file to copy it into (so that the operator's console and TLOG are not mutually locked out), copies TLOG into the temporary file, then closes, recreates and unlocks (via GFUNLOCK) TLOG. The operator's console is then locked via GFLOCK, the temporary file is copied onto it and the operator's console is unlocked via GFUNLOCK. SPOOL then begins another cycle of the infinite loop by sleeping for the requisite number of seconds. See figure 51 for data flow.

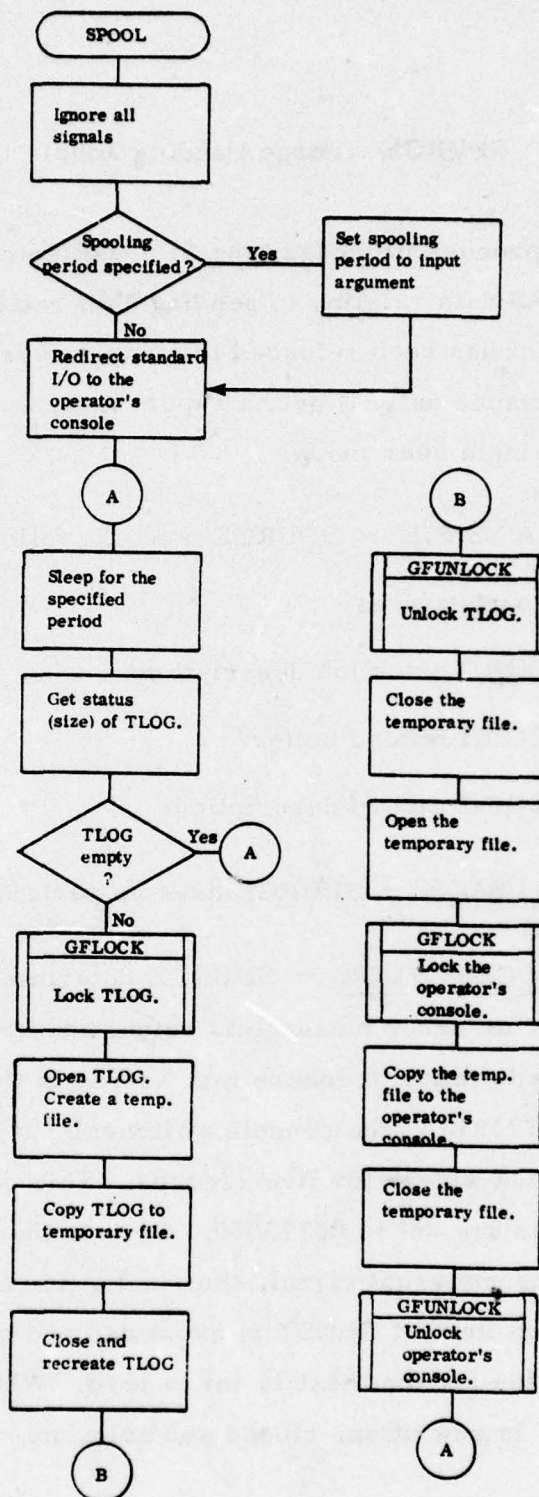


Figure 51. SPOOL Process Data Flow

SPURGE (Purge Pending Jobs)

The SPURGE process processes the SPURGE command. It is responsible for purging TAS data relating to pending TAS batch and interactive jobs after the file system has been reloaded following a system crash. This command may be issued only 1) by the super-user, and 2) after TAS has been booted up in single user mode.

GLOBAL DATA USAGE - SPURGE uses the following global data:

GHOST - host statuses
GJOBS - BQRD batch job descriptions
GLBUF - COINI record buffer
GLJOBS - COINI record descriptions

LOCAL DATA USAGE - SPURGE uses no pertinent local data.

GENERAL PROCESS FLOW - SPURGE determines if the user is the super-user. If not, an error message is output and SPURGE terminates. Otherwise, a check is made to ensure that TAS is in the single user mode (as indicated by 00773030 in the console switches). If it is not, a message is output and SPURGE sleeps for five seconds. This cycle is repeated until the console switches are set to 00773030. Once in the single user mode, SPURGE ignores the interrupt signal, then locks (via GFLOCK), opens and reads GHOST. Each item of GHOST is examined and if that host is batch, the number of queries for that host is set to zero. When all items have been processed, GHOST is rewritten, closed and unlocked via GFUNLOCK. Next

a loop is begun in which each batch host under TAS is processed. For each batch host, its GJOBS is locked via GFLOCK, GJOBS and GRESP are opened and a loop is begun in which all GJOBS items are processed. For each occupied GJOBS item, if the job is not an INTG, its data file is unlinked and the next item is processed. Otherwise, if the job is an INTG, and a response processor pathname is present, that pathname is read from GRESP and the response processor is called via a fork/execute sequence with a job status of "lost INTG". The data file for this GJOBS item is then unlinked, and the next GJOBS item is processed.

When all GJOBS items have been processed for a particular host, the GJOBS and GRESP files are recreated, GJOBS is closed and unlocked via GFUNLOCK and the next batch host is processed. When all batch hosts have been processed, GLJOBS is locked via GFLOCK and opened. For each occupied item in GLJOBS, the COINI record file it refers to is unlinked. GLJOBS is then recreated, closed and unlocked via GFUNLOCK. GLBUF is then locked via GFLOCK, recreated, unlocked via GFUNLOCK, and SPURGE terminates. See figure 52 for data flow.

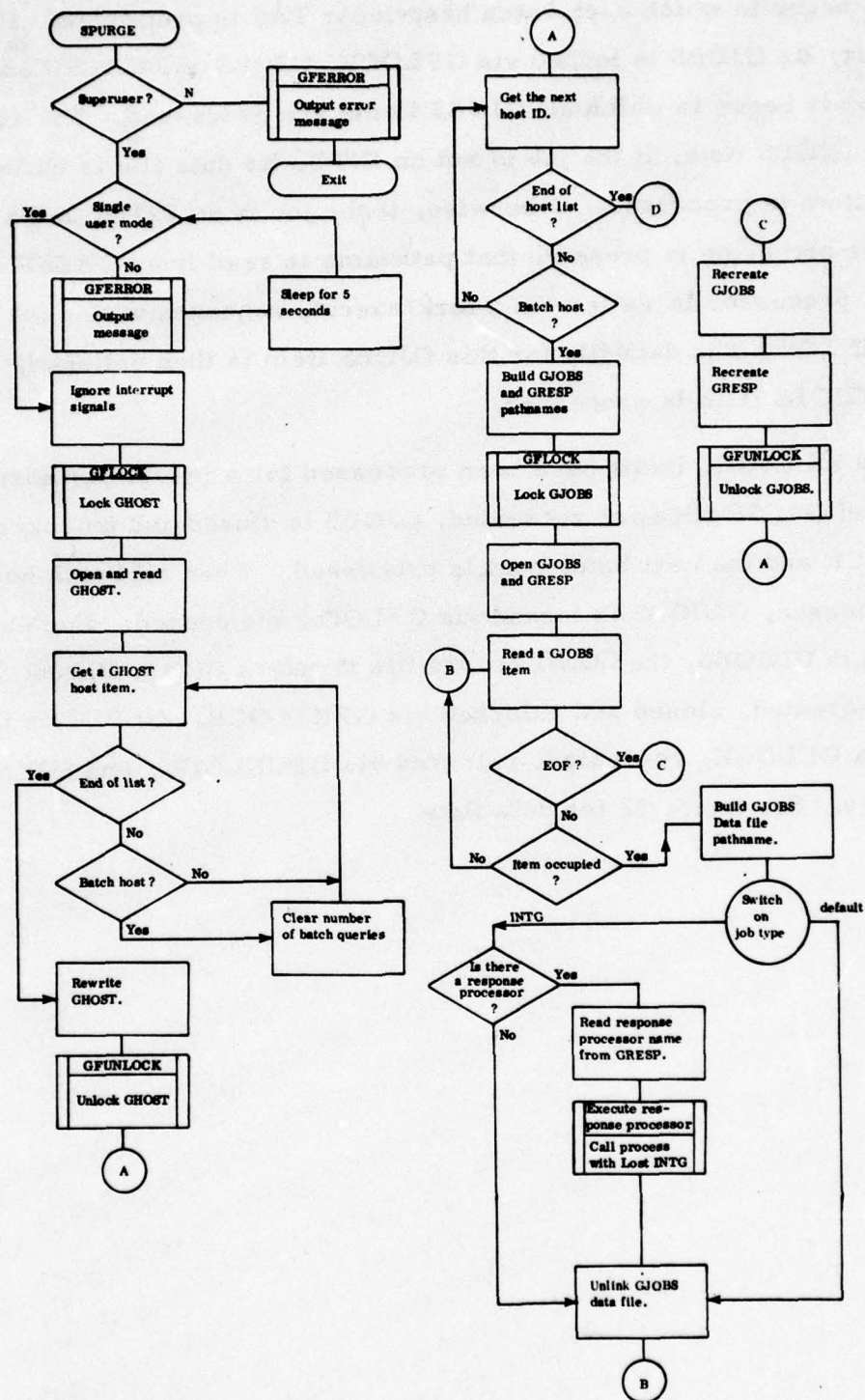


Figure 52. SPURGE Process Data Flow (Sheet 1 of 2)

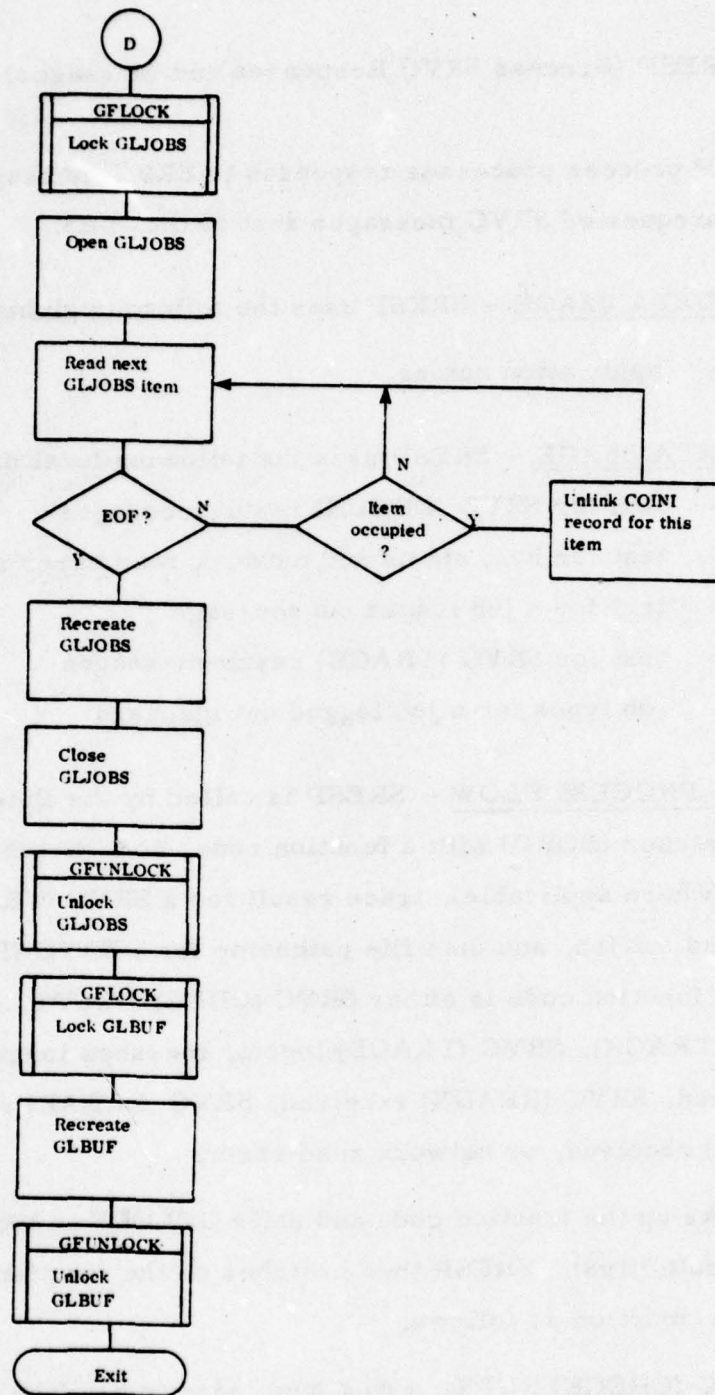


Figure 52. SPURGE Process Data Flow (Sheet 2 of 2)

SRESP (Process SRVC Responses and Messages)

The SRESP process processes responses to SRVC messages sent from the TAS and unrequested SRVC messages sent to the TAS.

GLOBAL DATA USAGE - SRESP uses the following global data:

GAPNM - application names

LOCAL DATA USAGE - SRESP uses the following local data:

SPCHK - text for SRVC (CHECK) result messages

SPHOST - text for host status and network read error messages

SPLOG - text for a job logged out message

SPMSG - text for SRVC (TRACE) result messages

SPTYP - job types for a job logged out message

GENERAL PROCESS FLOW - SRESP is called by the Batch Query and Response Dispatcher (BQRD) with a function code, host ID (where applicable), traced JOBID (where applicable), trace result for a SRVC (TRACK), job type for a logged out job, and data file pathname for a SRVC (PRINT) received. The function code is either SRVC (CHECK) RCPT, SRVC (CHECK) logout, SRVC (TRACK), SRVC (TRACE) logout, message logged out, SRVC (PRINT) received, SRVC (READY) received, SRVC (BREAK) received, SRVC (ALARM) received, or network read error.

SRESP picks up the function code and calls GFLOCK to lock the operator's console (tty8). SRESP then switches on the function code and processes each function as follows.

For a SRVC (CHECK) RCPT, a flag (and index to SPCHK) is set to one. For either SRVC (CHECK) RCPT or SRVC (CHECK) logout (where the flag is zero by default), the appropriate message from SPCHK is output to tty8

followed by the host name, the current time and a blank line. SPERROR is then called to exit.

For a job logged out, the job type from SPTYP and the logged out message in SPLOG are output to tty8 followed by the host name, current time and a blank line. SPERROR is then called to exit.

For a SRVC (TRACK), a flag (and index to SPMSG) is set to one, and the trace result is stored in the SRVC (TRACK) message in SPMSG. For either SRVC (TRACK) or SRVC (TRACE) logout (where the flag is zero by default), the traced JOBID is stored in the appropriate message, the message in SPMSG is output to tty8, followed by the current time and a blank line. SPERROR is then called to exit.

For either SRVC READY, BREAK or ALARM received, or a network read error, the appropriate message in SPHOST is output to tty8, followed by the host name, current time and a blank line. SPERROR is then called to exit.

For a SRVC (PRINT) received, a SRVC (PRINT) message is output to tty8 followed by the host name, current time and a blank line. The data file containing the PRINT data is opened and read and the data is output to tty8. SPERROR is then called to exit. See figure 53 for data flow.

MAJOR FUNCTION DESCRIPTIONS

SPERROR (Output Error and/or Exit) - SPERROR is called with an error number or zero. If tty8 is locked, GFUNLOCK is called to unlock tty8. If an error number is specified, GFERROR is called to output the error. SPERROR then exits.

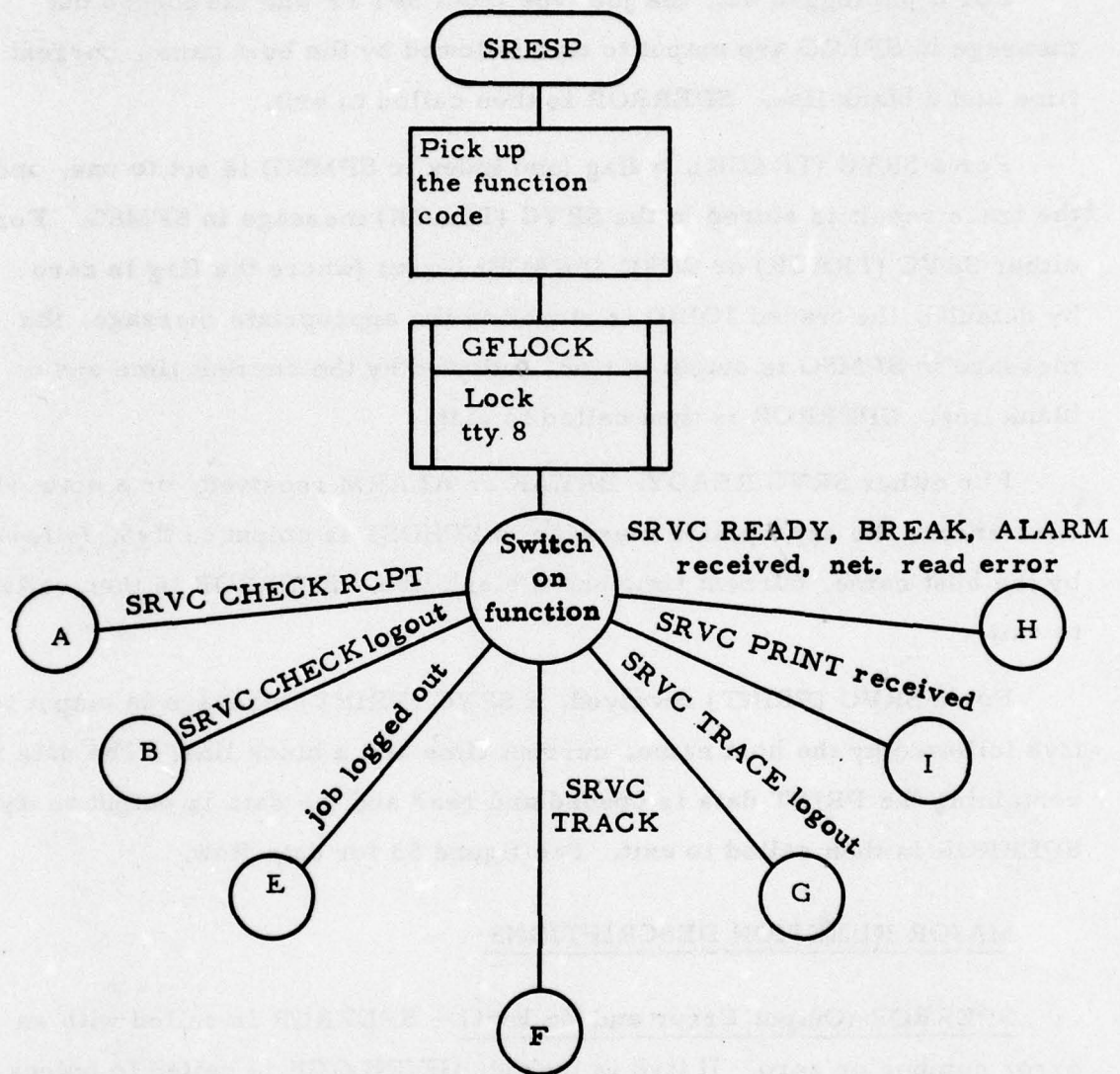


Figure 53. SRESP Process Data Flow (Sheet 1 of 3)

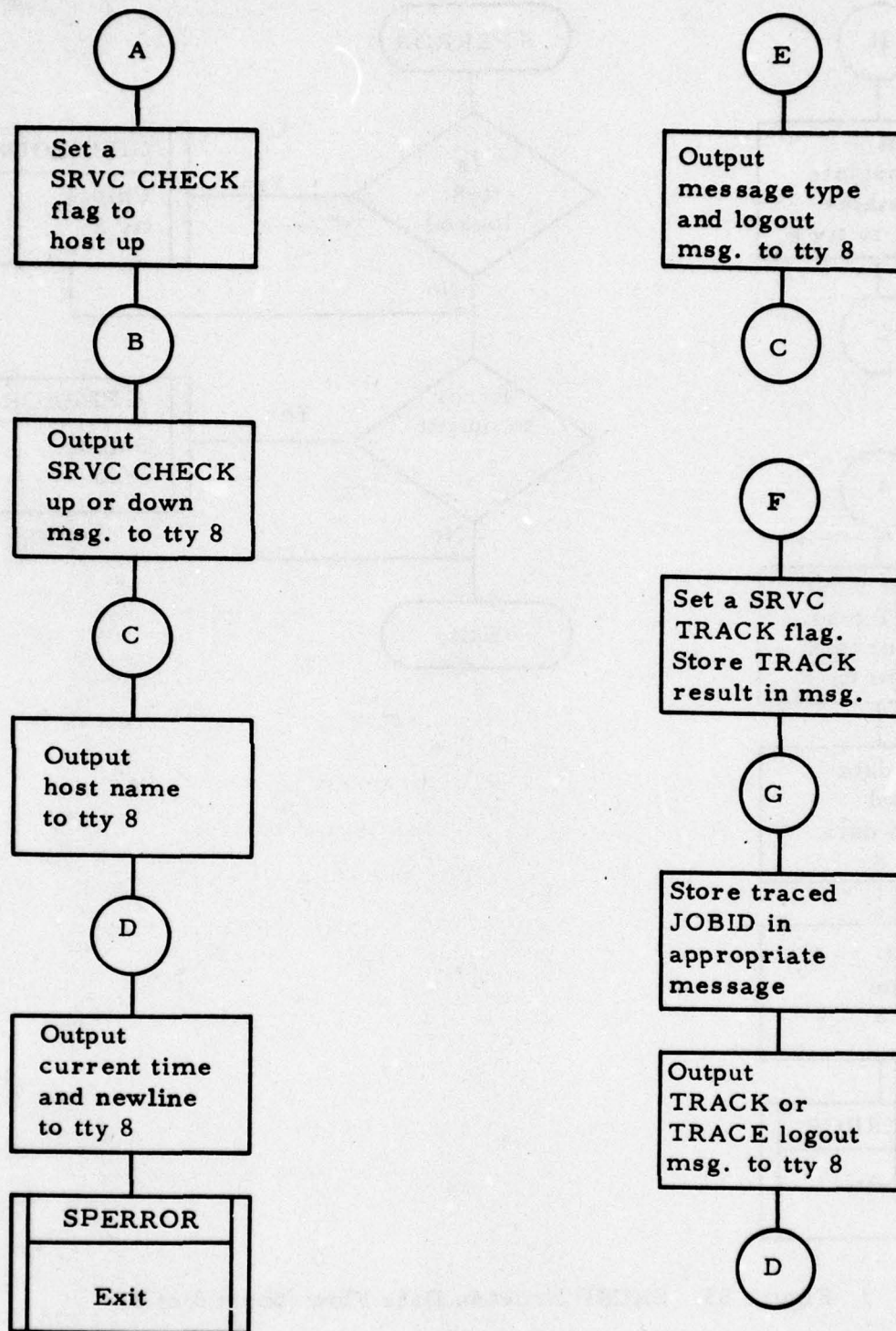


Figure 53. SRESP Process Data Flow (Sheet 2 of 3)

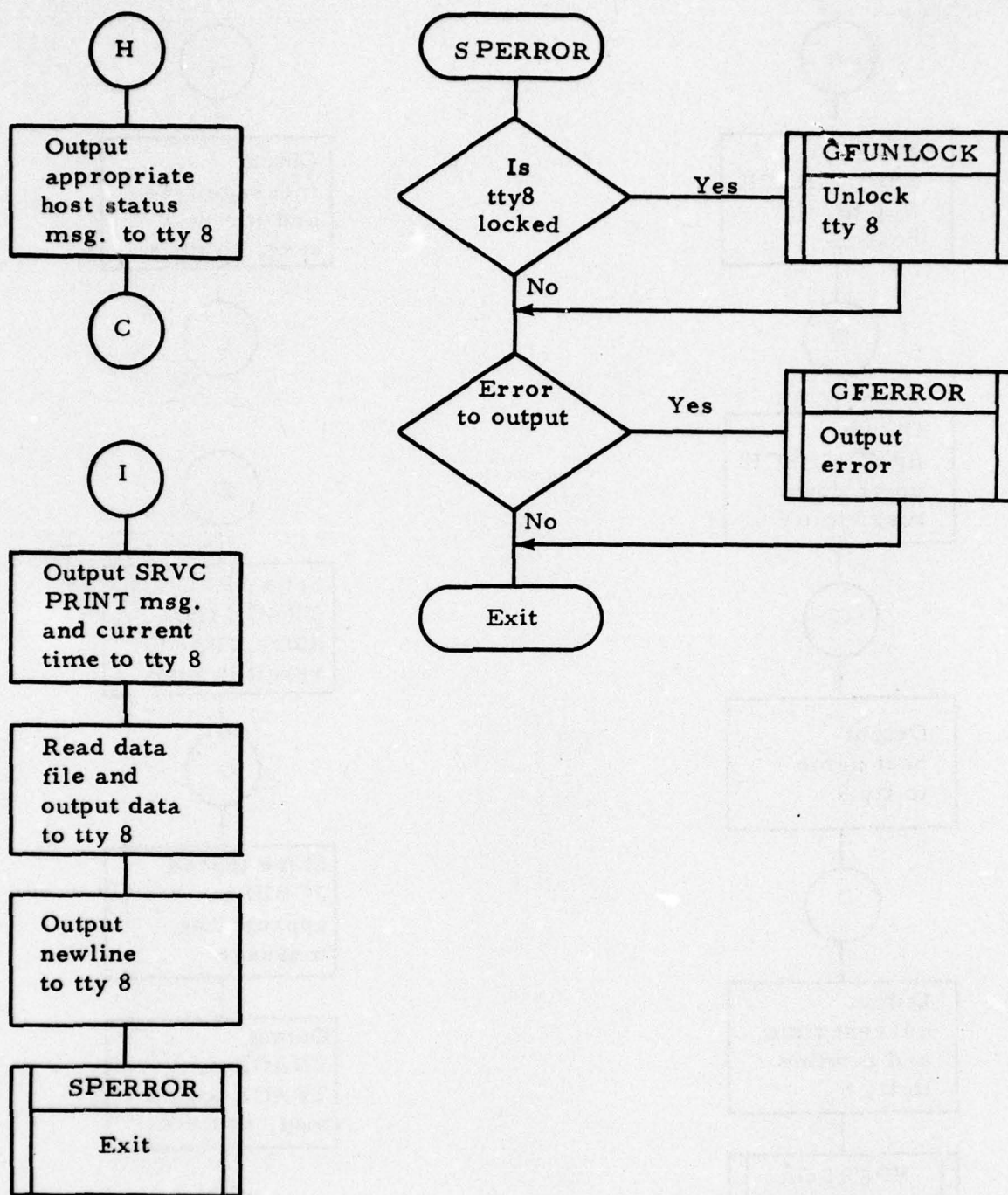


Figure 53. SRESP Process Data Flow (Sheet 3 of 3)

SRVC (Process a SRVC Request)

The SRVC process processes the SRVC command. It is responsible for handling all SRVC requests and is restricted to the TASMASTER.

GLOBAL DATA USAGE - SRVC uses the following global data:

GHOST - host statuses

GFILE - user text file descriptions

LOCAL DATA USAGE - SRVC uses the following local data:

Temporary user text file.

GENERAL PROCESS FLOW - SRVC first checks to see if the call is an internal call. If it is, an internal call flag is set. Otherwise, a test is made to ensure that the user is the TASMASTER. If not, an error message is output and SRVC terminates. Next, if at least two arguments are not present in the call, an error message is output and SRVC terminates.

The SRVC subtype is then capitalized and checked against a list of legal subtypes. If it is not valid, an error message is output and SRVC terminates. CHOST is then opened, read, and closed, and a test is made to determine if "all hosts" option (for all TAS hosts) was specified in the argument list. If so, an all hosts flag is set; otherwise, the host name is picked up and capitalized. If the host name is not valid, an error message is output and SRVC terminates.

If the function is SRVC TRACE, a test is made for a JOBID input argument. If a JOBID argument was input, a series of tests are made to validate the request. If the specified host is not up, an error message is output and SRVC

terminates. If the specified host is not batch, an error message is output and SRVC terminates. If the JOBID is not the correct length, an error message is output and SRVC terminates. If all these tests are passed, a temporary JOBID file is created, the JOBID is written and the file is closed. The response processor and data file pathnames are then set up together with the JOBID (via GFJOBID), and the BQRD process is called with the specified SRVC request. Whether or not the SRVC TRACE was sent, a message is output before SRVC terminates.

If the function was not TRACE or there was no JOBID input, SRVC checks to see if the function is SRVC PRINT. If so, a test is made to determine if the SRVC PRINT data is to come from a user's file. If a user file is specified, GFILE is opened and item zero is read. If the user has no files, an error message is output and SRVC terminates. Otherwise, the file count is picked up and GFILE is searched for a match to the specified user file name. If no match is found, an error message is output and SRVC terminates. Otherwise, GFILE is closed, interrupt signals are ignored and a link is created to the user's text file.

If the SRVC PRINT data is not to come from a user's file, a temporary text file is created, and the user is instructed to input the desired PRINT data at the keyboard, terminating the input with an EOT (control-d) character. The data from the keyboard is then copied to the temporary file until an end-of-text is read.

At this point the data file pathname for the SRVC PRINT is set up.

If the function was not SRVC PRINT, a test is made for a function of SRVC CHECK. If the function is CHECK, the response processor and data file pathnames are set up.

A loop is then performed over all hosts, unless the "all hosts" option was not specified. For each host, or the specified host if "all" was not specified, a check is made to ensure that the host is batch. If it is not, but the "all hosts" option was specified, the next host is processed; otherwise, an error message is output and SRVC terminates.

If the host is batch, BQRD is called with the specified SRVC function. Whether or not the SRVC message was sent, a message is output to the user.

If the "all hosts" option was not specified, an end message is output and SRVC terminates. Otherwise, if more hosts are to be processed, the loop is repeated.

When all hosts have been processed, an end message is output and SRVC terminates. See figure 54 for data flow.

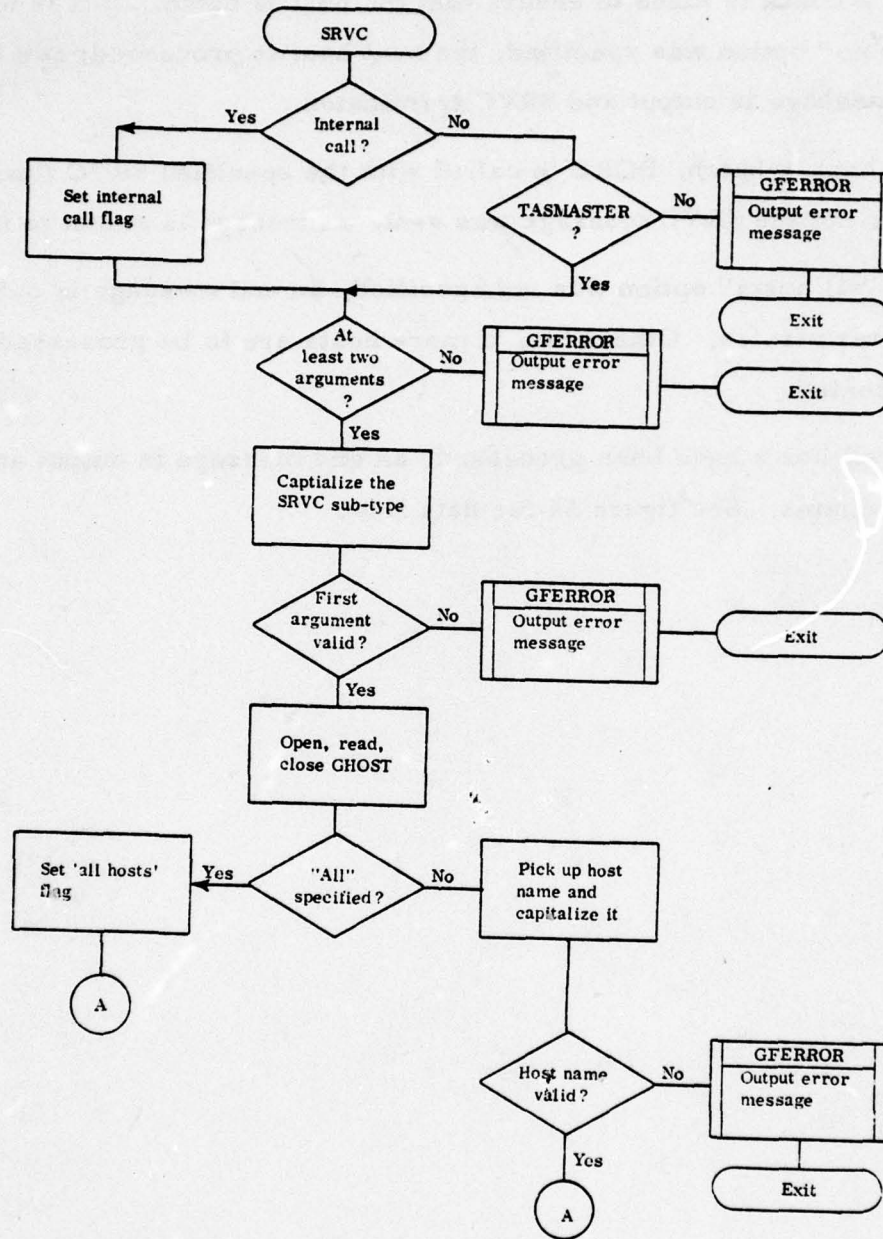


Figure 54. SRVC Process Data Flow (Sheet 1 of 4)

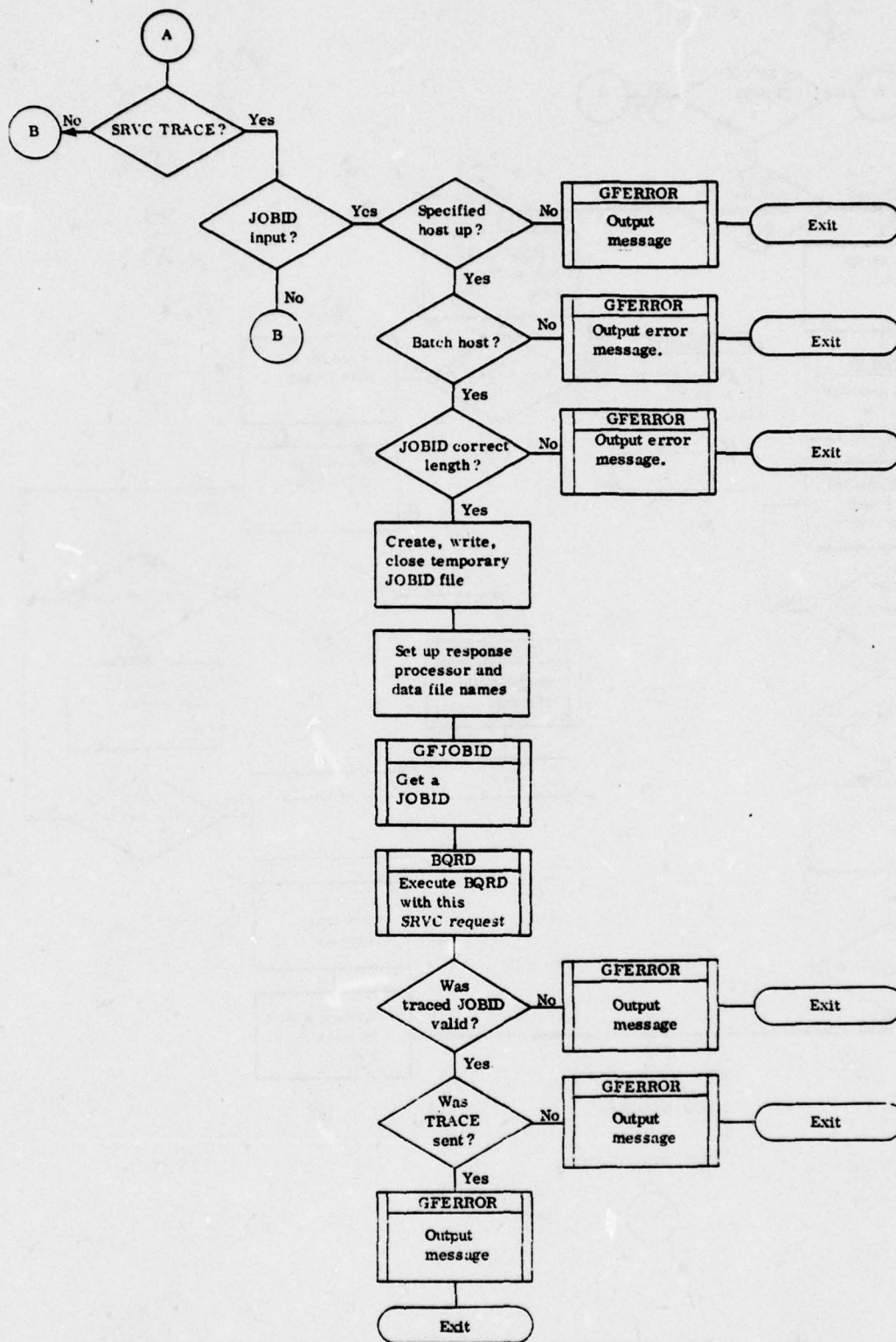


Figure 54. SRVC Process Data Flow (Sheet 2 of 4)

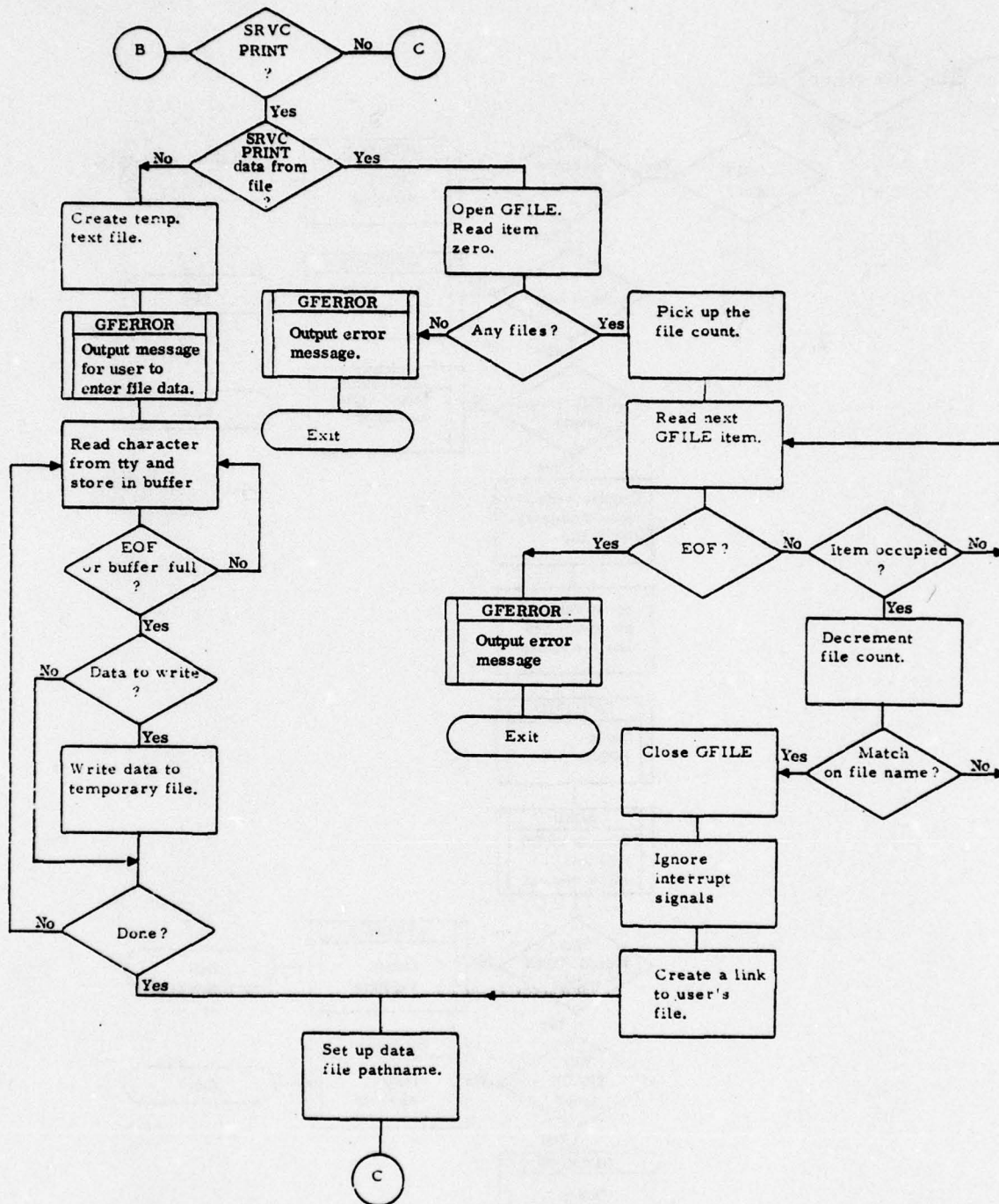


Figure 54. SRVC Process Data Flow (Sheet 3 of 4)

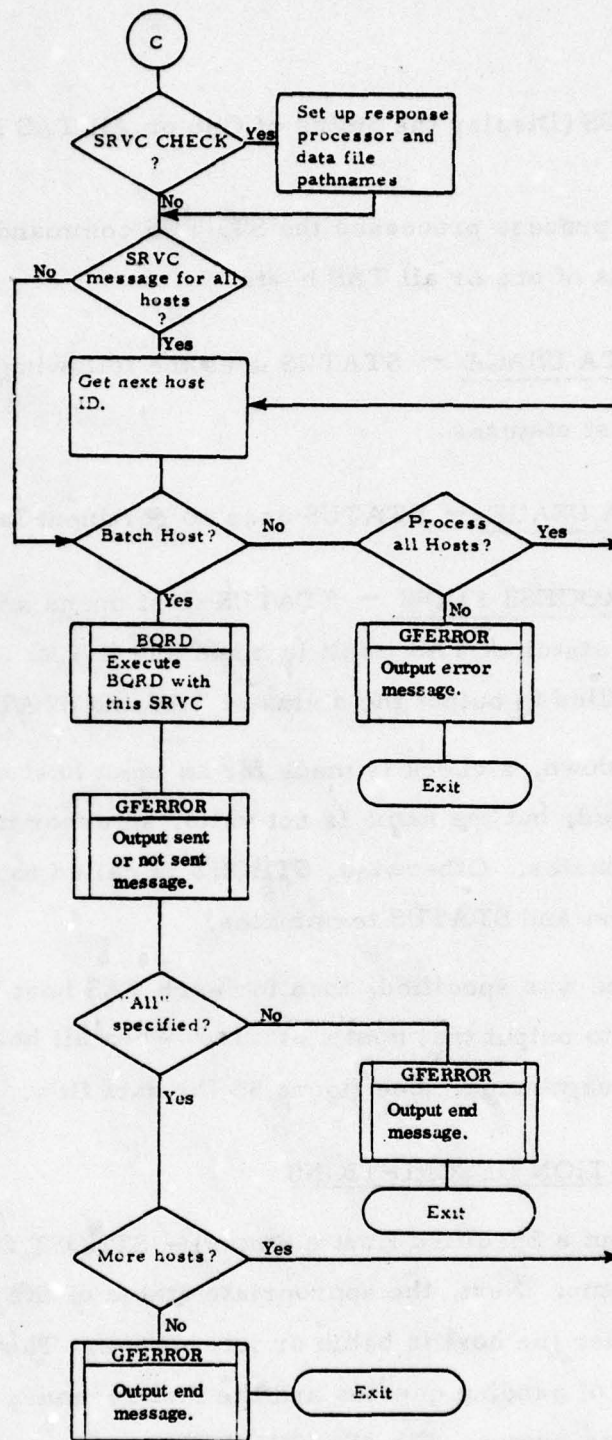


Figure 54. SRVC Process Data Flow (Sheet 4 of 4)

STATUS (Display the Status of One or All TAS Hosts)

The STATUS process processes the STATUS command. It allows a user to obtain the status of one or all TAS hosts.

GLOBAL DATA USAGE - STATUS uses the following global data:

GHOST - host statuses

LOCAL DATA USAGE - STATUS uses no pertinent local data.

GENERAL PROCESS FLOW - STATUS first opens and reads GHOST. Then a test of the status of TAS itself is made and if TAS is down the function STHOST is called to output the status of TAS and STATUS terminates.

If TAS is not down, a check is made for an input host name argument. If a host is specified, but the name is not valid, an error message is output and STATUS terminates. Otherwise, STHOST is called to output the status of the specified host and STATUS terminates.

If no host name was specified, then for each TAS host listed in GHOST, STHOST is called to output that host's status. When all hosts have been processed, STATUS terminates. See figure 55 for data flow.

MAJOR FUNCTION DESCRIPTIONS

STHOST (Output a Specified Host's Status) - STHOST first outputs the specified host's name. Next, the appropriate status of the host is output, dependent on whether the host is batch or interactive. Then, if the host is batch, the number of pending queries and the last response and submission times of a query are output. Finally, STHOST returns.

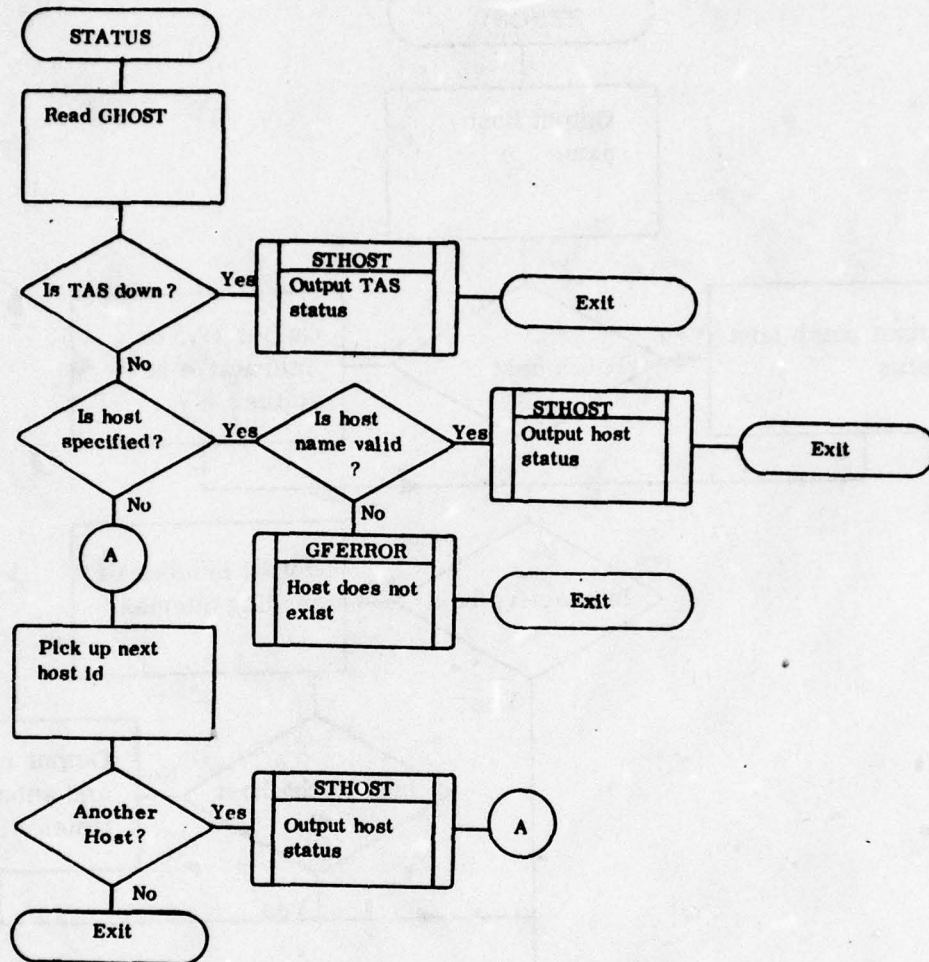


Figure 55. STATUS Process Data Flow (Sheet 1 of 2)

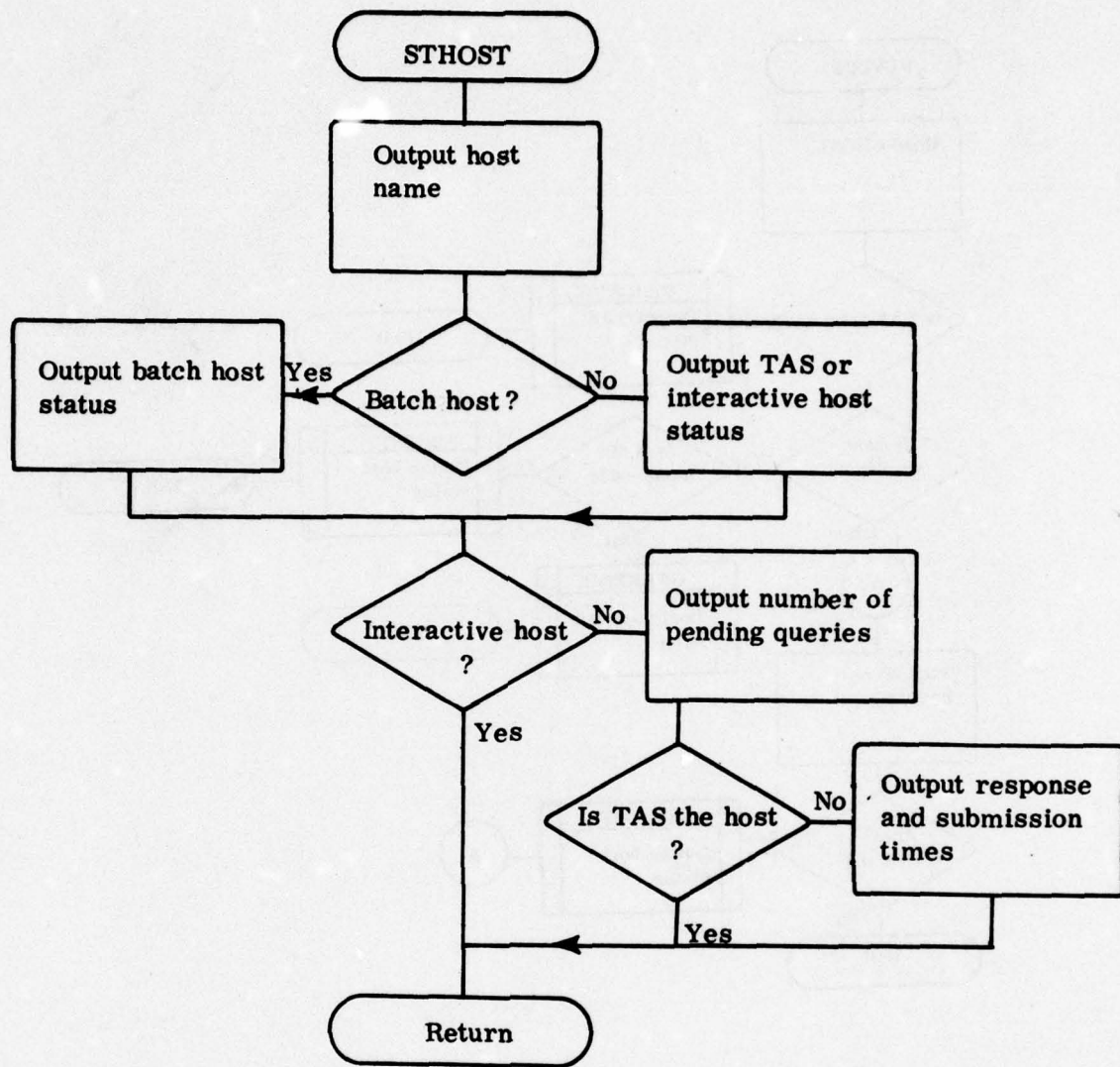


Figure 55. STATUS Process Data Flow (Sheet 2 of 2)

SYSRES (Calculate System Resources)

The SYSRES process can be activated either as a daemon when the system is booted up (in "etc/rc") or as a TAS command by the TASMASTER. SYSRES calculates and, if applicable, displays the percentage of used space for each of the TAS system's major storage devices.

GLOBAL DATA USAGE - Since SYSRES is basically a system program, the standard TAS global data structures are not pertinent to SYSRES. However, TAS global DEFINESs are used by the SYSRES process.

LOCAL DATA USAGE - SYSRES uses the following local data structures:

SBLOCK - super block structure
DARGV - array for storage device pathnames

GENERAL PROCESS FLOW - If the SYSRES process is being activated by a user other than the TASMASTER, an error is output and SYSRES terminates. If the SYSRES process is being called to act as a daemon, the percentage input parameter is retrieved and stored. Then the daemon SYSRES sleeps for its indicated length of time.

Once the SYSRES daemon has awakened or the SYSRES command has finished validating the user, the storage file systems indicated in array DARGV are individually scanned by the function CHECK. CHECK determines for a given file system the number of used blocks of storage and then calculates the percentage of used storage space. CHECK, combined with the functions it calls upon to aid in the calculations, forms an adapted version of the UNIX command ICHECK.

After each file structure is evaluated for used storage, the process either exits if the process is in the command mode or sleeps for the allotted length of time if the process is a daemon. See figure 56 for data flow.

MAJOR FUNCTION DESCRIPTIONS

CHECK — The CHECK function stabilizes the TAS environment by invoking the UNIX SYNC system call, reads in the super block for the desired file systems and determines the number of blocks devoted to the i-list. Each i-list block is read and processed to determine the classifications of blocks in the file system, to find possible errors, and to count the number of used blocks. The CHECK function again performs the SYNC system call, reads the super block into SBLOCK and this time determines the number of free blocks by a series of ALLOC calls. Given the number of used and free blocks, the percentage of used storage space on the given file system is calculated.

If the process is a daemon, the percentage is output on the operator's console only if the percentage is equal to or greater than the percentage input parameter. In order to output data on the operator's console (tty-8), SYSRES forks to itself and the child process sets the user ID to the TAS ID (TUID), locks the operator's console via the GFLOCK function, displays the percentage, unlocks the operator's console via the GFUNLOCK function and exits. The parent awaits the return from the child process. However, if the process is in the command mode, the percentage of used disk space for the specific file system is output. In either situation, CHECK returns.

PASS1 (Determine Inode Block Type) — The function PASS1 determines the inode block type and calls function CHK with the address of the inode and the inode block type.

CHK (Counter for Used Inodes) — The function CHK counts the used inodes. If the inode block number is out-of-range, a non-zero value is returned by CHK; otherwise, the value zero is returned.

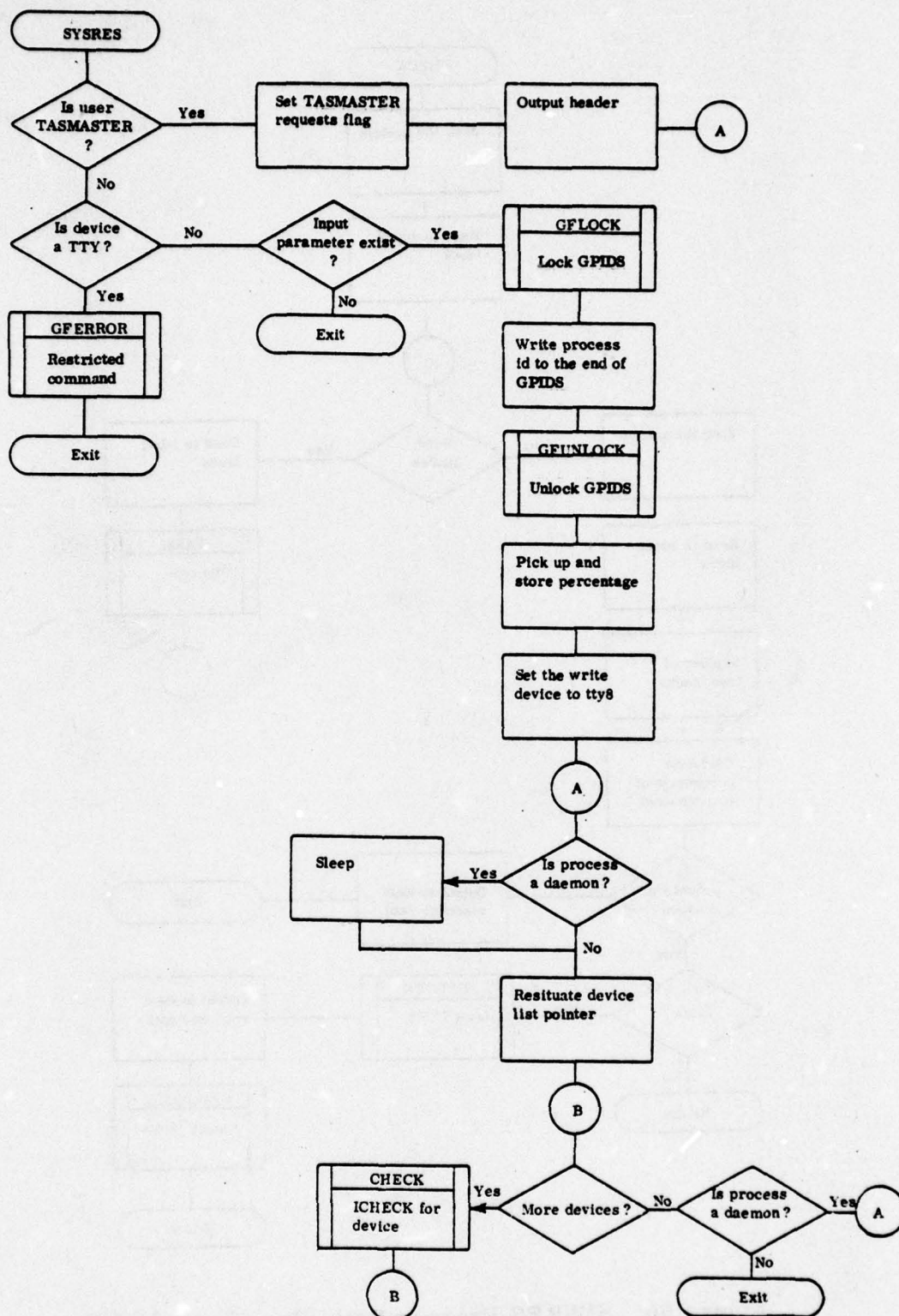


Figure 56. SYSRES Process Data Flow (Sheet 1 of 4)

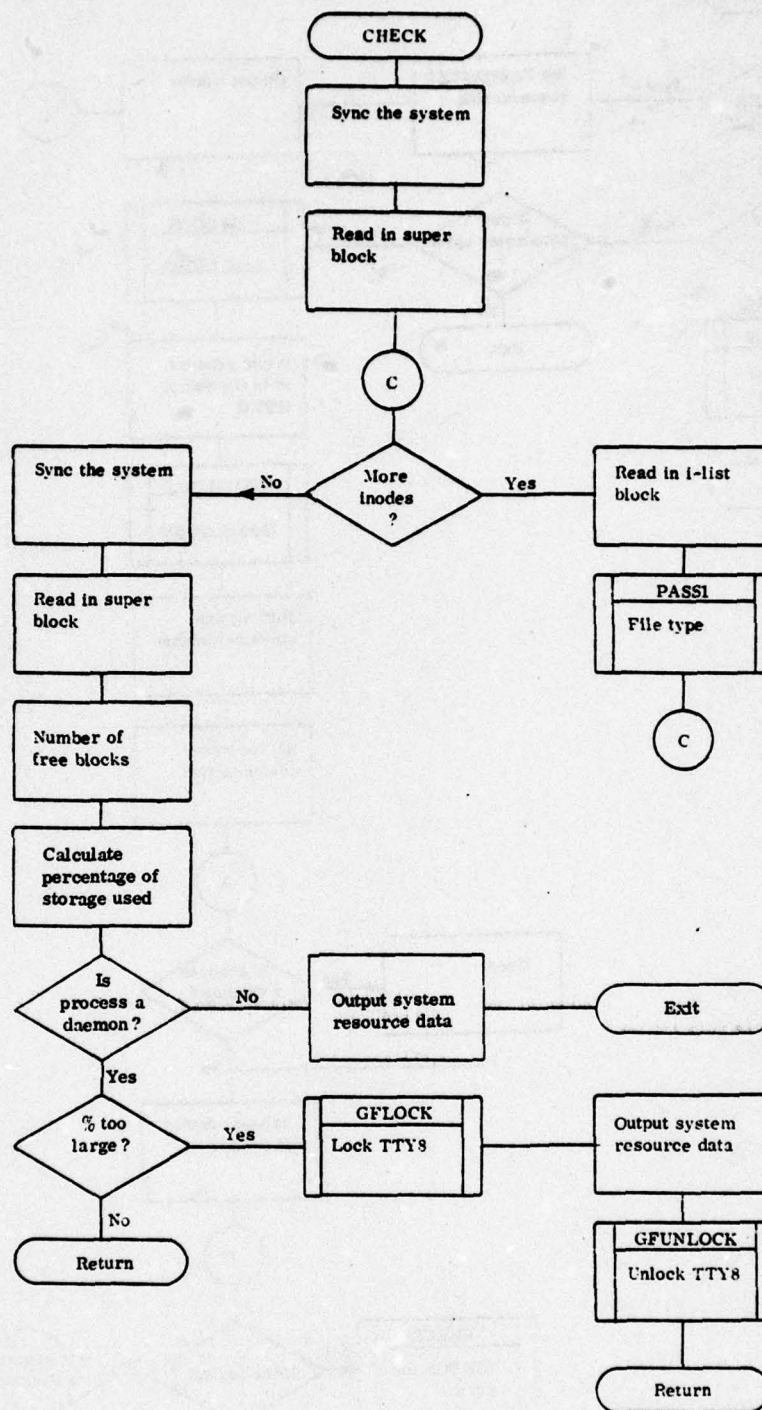


Figure 56. SYSRES Process Data Flow (Sheet 2 of 4)

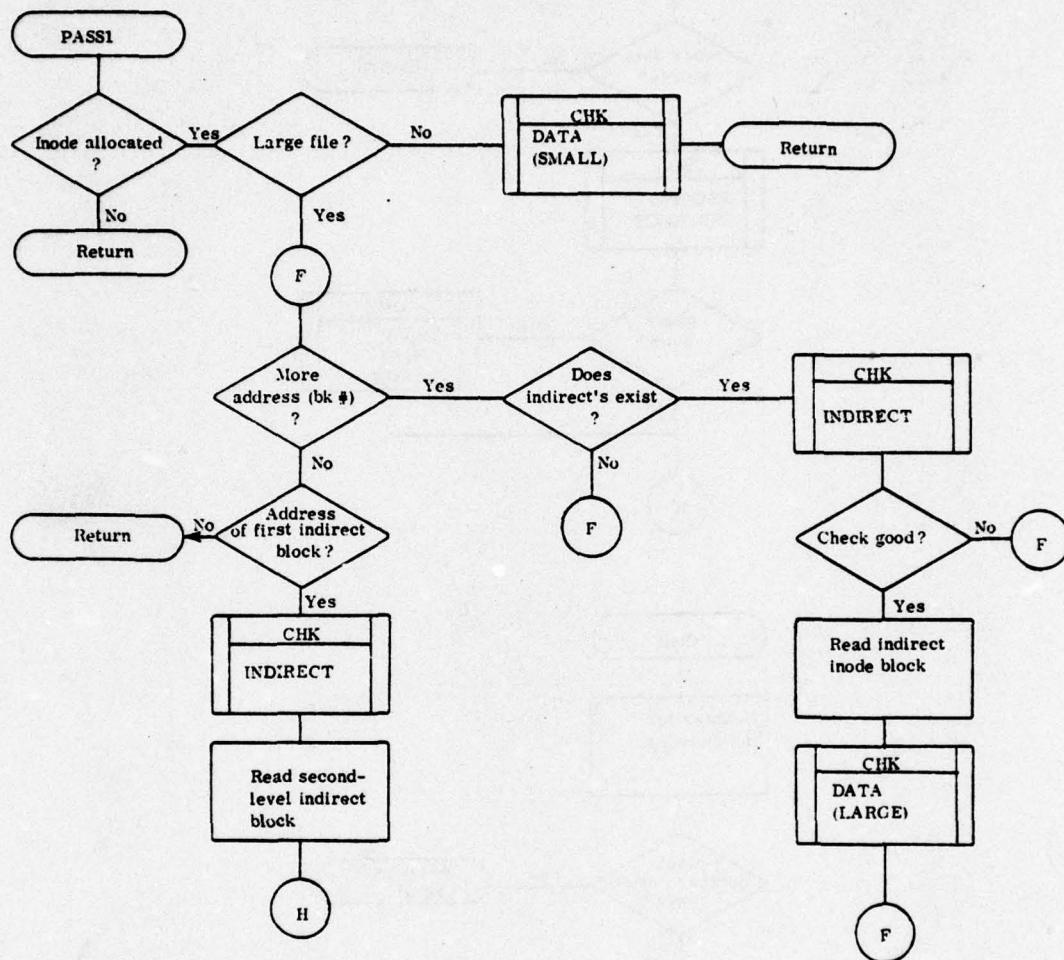


Figure 56. SYSRES Process Data Flow (Sheet 3 of 4)

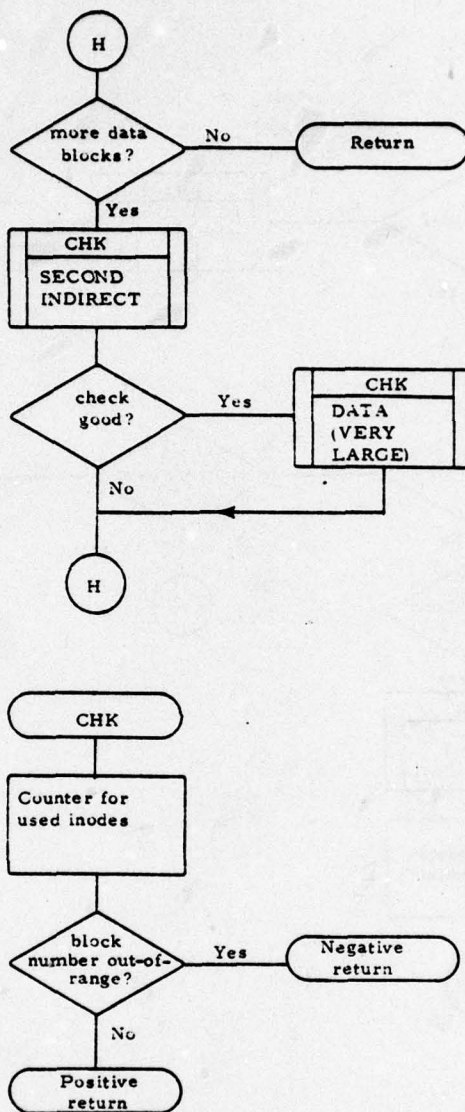


Figure 56. SYSRES Process Data Flow (Sheet 4 of 4)

SYSTAT (Display Batch Job Statuses)

The SYSTAT process processes the SYSTAT command. It allows the TASMASTER to obtain a listing of the status of each batch job in TAS for a specified host or all hosts.

GLOBAL DATA USAGE - SYSTAT uses the following global data:

GHOST - host statuses
GJOBS - BQRD batch job descriptions
GRES P - batch query response processor pathnames
GAPNM - application names

LOCAL DATA USAGE - SYSTAT uses no pertinent local data.

GENERAL PROCESS FLOW - SYSTAT first checks to see if the user is the TASMASTER. If not, an error message is output and SYSTAT terminates. Otherwise, GHOST is opened, read and closed.

A check is then made for an input argument. If none is given, an "all hosts" flag is cleared. Otherwise, the argument is capitalized and validated against a list of application names in GAPNM. If the name is not found, an error message is output and SYSTAT terminates. If a match is found, but the specified host is not batch, an error message is output and SYSTAT terminates.

The current time is saved and a header is output.

For each TAS host (if no host was specified) or for the host specified by the input argument, the function SYHOST is called to output the batch job statuses for that host. When all hosts have been processed, SYSTAT terminates. See figure 57 for data flow.

MAJOR FUNCTION DESCRIPTIONS

SYHOST (Output a Host's Batch Job Statuses) — SYHOST first determines if the host specified in the call is batch. If not, a message is output and SYHOST returns. If the host is batch, a header for the host output is output. Next, GJOBS is opened and item zero is read. For each occupied item in GJOBS, the status data for the referenced batch job is output. When the end-of-file is read, a check is made to determine if any jobs were found. If not, a message is output. Finally, SYSHOST returns.

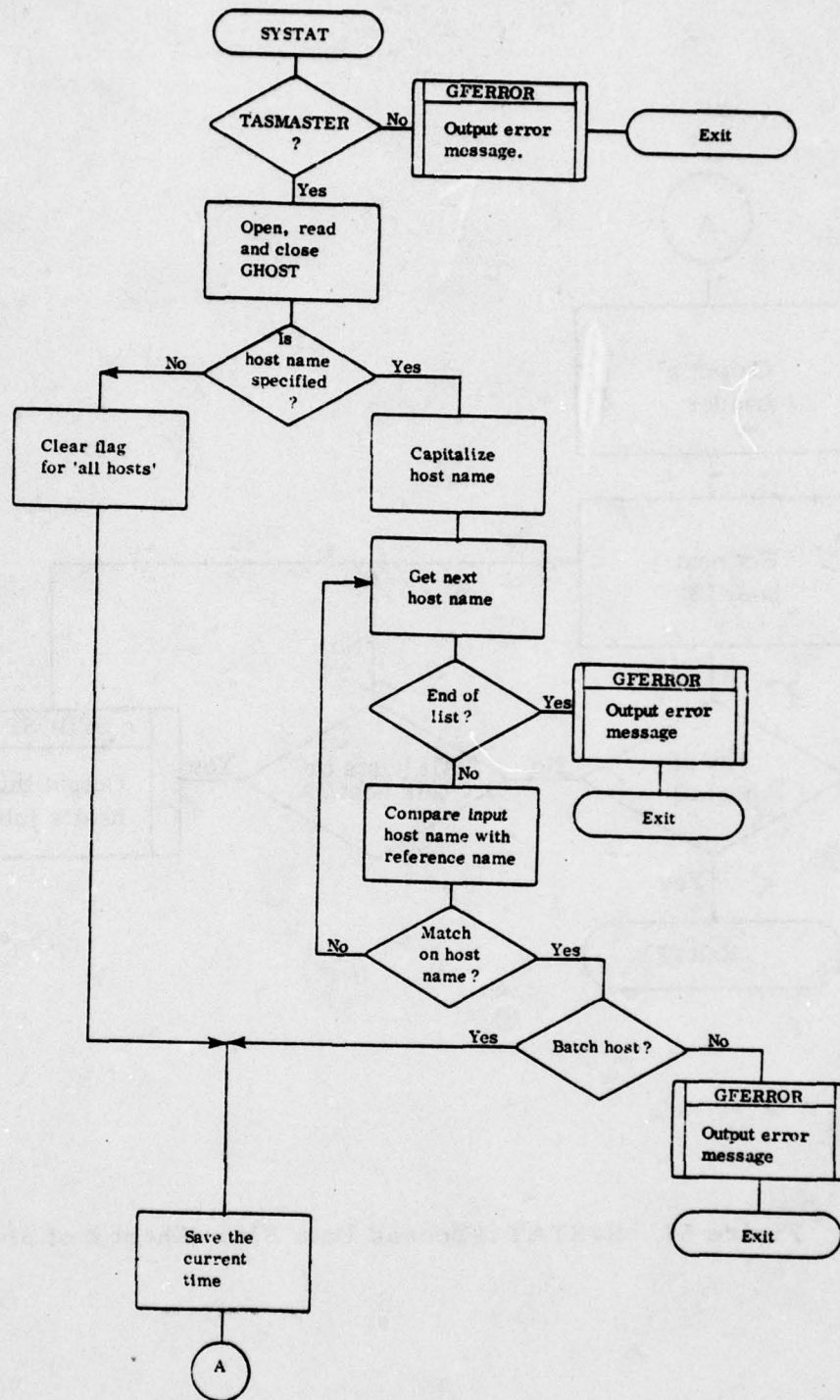


Figure 57. SYSTAT Process Data Flow (Sheet 1 of 3)

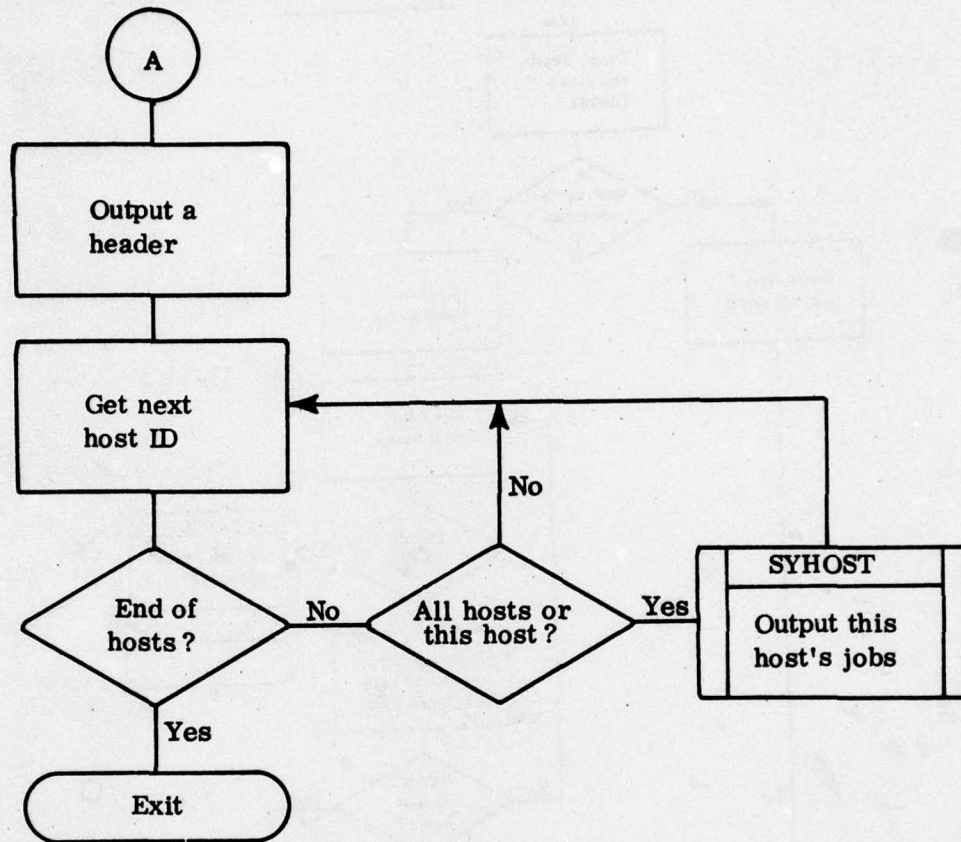


Figure 57. SYSTAT Process Data Flow (Sheet 2 of 3)

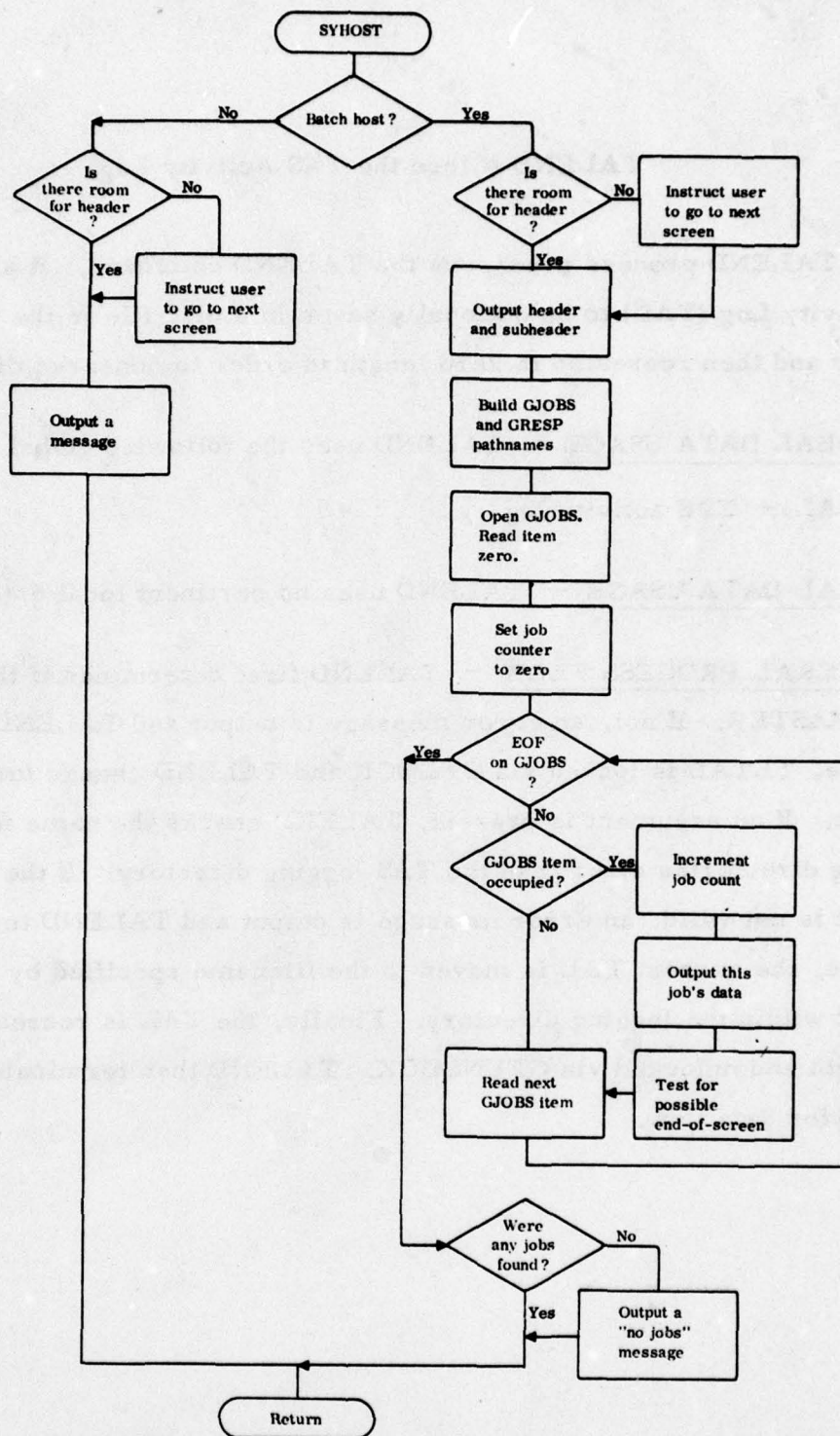


Figure 57. SYSTAT Process Data Flow (Sheet 3 of 3)

TALEND (Close the TAS Activity Log)

The TALEND process processes the TALEND command. It allows the TAS Activity Log (TAL) to be optionally saved in a disk file in the TAS logging directory and then recreated to zero length in order to conserve disk space.

GLOBAL DATA USAGE - TALEND uses the following global data:

TLTAL - TAS activity log

LOCAL DATA USAGE - TALEND uses no pertinent local data.

GENERAL PROCESS FLOW - TALEND first determines if the user is the TASMMASTER. If not, an error message is output and TALEND exits. Otherwise, TLTAL is locked via GFLOCK and TALEND checks for an input argument. If an argument is present, TALEND checks the name for slashes indicating directories other than the TAS logging directory. If the filename argument is not valid, an error message is output and TALEND terminates. Otherwise, the current TAL is moved to the filename specified by the input argument within the logging directory. Finally, the TAL is recreated to zero length and unlocked via GFUNLOCK. TALEND then terminates. See figure 58 for data flow.

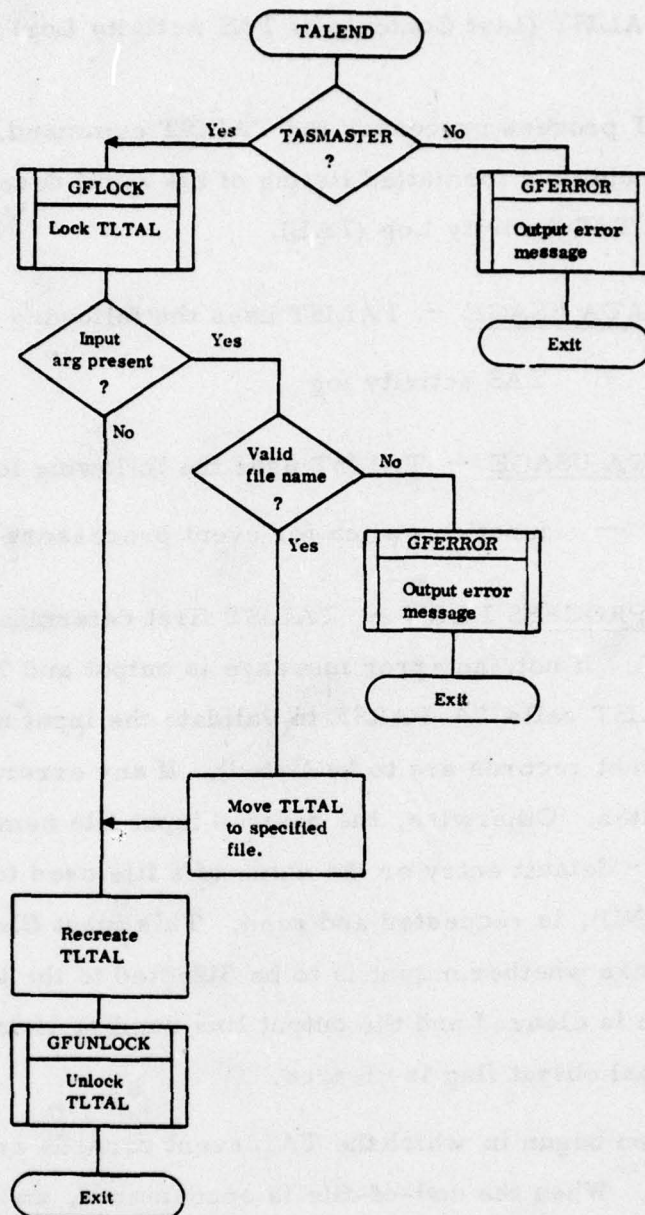


Figure 58. TALEND Process Data Flow

TALIST (List Contents of TAS Activity Log)

The TALIST process processes the TALIST command. It allows the TASMMASTER to obtain a formatted listing of the event description records contained in the TAS Activity Log (TAL).

GLOBAL DATA USAGE - TALIST uses the following global data:

TLTAL - TAS activity log

LOCAL DATA USAGE - TALIST uses the following local data:

ta-funct - function switch for event processors

GENERAL PROCESS FLOW - TALIST first determines if the user is the TASMMASTER. If not, an error message is output and TALIST terminates. Otherwise, TALIST calls TA-PARSE to validate the input arguments (that specify which event records are to be listed). If any errors are encountered, TALIST terminates. Otherwise, the desired input file name, either the current TAL as a default entry or the name of a file used to save a previous TAL (with TALEND), is requested and read. This input file is then opened and TALIST checks whether output is to be directed to the terminal screen. If so, the screen is cleared and the output line number is initialized; otherwise, the terminal output flag is cleared.

A loop is then begun in which the TAL event records are read sequentially until end-of-file. When the end-of-file is encountered, an "end of data" message is output and TALIST terminates. Otherwise, if the event code is valid, the appropriate function is called (via a function switch ta-funct

containing the addresses of the functions TA-00 - TA-13) to process (if necessary) that event record. The next event record is then read and the process is repeated. If, however, the event code for a given record is not valid, TALIST outputs an error message and terminates. See figure 59 for data flow.

MAJOR FUNCTION DESCRIPTIONS

TA-PARSE (Parse and Validate Input Arguments) - TA-PARSE parses and validates the input arguments to TALIST which specify those event records to be displayed. It converts those arguments to an internal form which allows them to be more readily used by the event processing functions.

TA-PARSE first sets the default search mode to "all events". If any arguments are present in the command line, the first one is validated against a list of search category keywords by TA-LOOKUP. If the argument is not a digit string or the word START (denoting a time-range specifier) then the keyword is processed according to the following four cases:

1. If the keyword is USER, a check is made to ensure that a user sni has been entered as the next argument. If not, an error message is output and TA-PARSE returns an error status. Otherwise, the user sni is saved, the argument pointer is moved to the next argument and the search mode is set to USER.
2. If the keyword is JOB, a check is made to ensure that a JOBID has been entered as the next argument. If not, an error message is output and TA-PARSE returns an error status. Otherwise, the JOBID is saved, the argument pointer is advanced and the search mode is set to JOBID.

3. If the keyword is VIO or LOG, the search mode is set to VIO or LOG, respectively.
4. If the keyword is not one of the above, an error message is output and TA-PARSE returns an error status.

Following processing of the search category keyword and its argument (if present), flags are set indicating the default condition that no time-range specifier has been entered. If there are any remaining arguments, the first is validated by TA-LOOKUP to see if it is START. If it is not, the start-time-specified flag is set and the argument is converted to double-precision integer format. If errors in conversion were encountered, an error message is output and TA-PARSE returns an error status. Otherwise, a check is made to determine if another argument (end-time specifier) is present. If it is not, an error message is output and TA-PARSE returns an error status. Otherwise, if the argument is not END, the end-time-specified flag is set and this time specifier is converted. If errors in conversion were encountered, an error message is output and TA-PARSE returns an error status. If the conversion was valid, and both start and end times were explicitly specified, a test is made to ensure that the end-time is greater than the start-time. If not, an error message is output and TA-PARSE returns an error status. Otherwise, TA-PARSE returns a no error status.

TA-LOOKUP (Find Argument in a Keyword Table) - TA-LOOKUP performs all keyword validation for TALIST. The input argument to be validated is first converted to upper case and then successively compared to each keyword in the reference table. If a match is found, the index of that keyword is returned; otherwise an error of -1 is returned.

TA-00 - TA-13 (Process Event Records) - The thirteen functions which process their respective event records from the TAL all have a common logic structure which is varied only as a function of the contents (i.e., data type) of the specific records. The purpose of these functions is to display the contents of a given TAL record if some portion of the record satisfies the search criteria. First a check is made to see if the time of the event record lies between the specified start and end times for the search (if given). If the record time is not within the time range, the function returns. If the search mode indicates that all record types are to be displayed, then the event record is formatted and output and the function returns. Otherwise, a series of tests are performed to determine if the event record matches the search parameters as described below:

1. If the record contains a user's sni, and the search mode is USER, then the record is formatted and output only if the input user sni matches the sni in the event record.
2. If the record contains a TAS JOBID, and the search mode is JOBID, then the record is formatted and output only if the input JOBID matches the JOBID in the event record.
3. If the event is a TAS LOGON or LOGOFF and the search mode is LOG, then the record will be formatted and output.
4. If the event implies a security violation and the search mode is VIO, then the record will be formatted and output.

If the above conditions are not met for the individual event records, the respective function simply returns. Otherwise, a return is made after the record has been formatted and output.

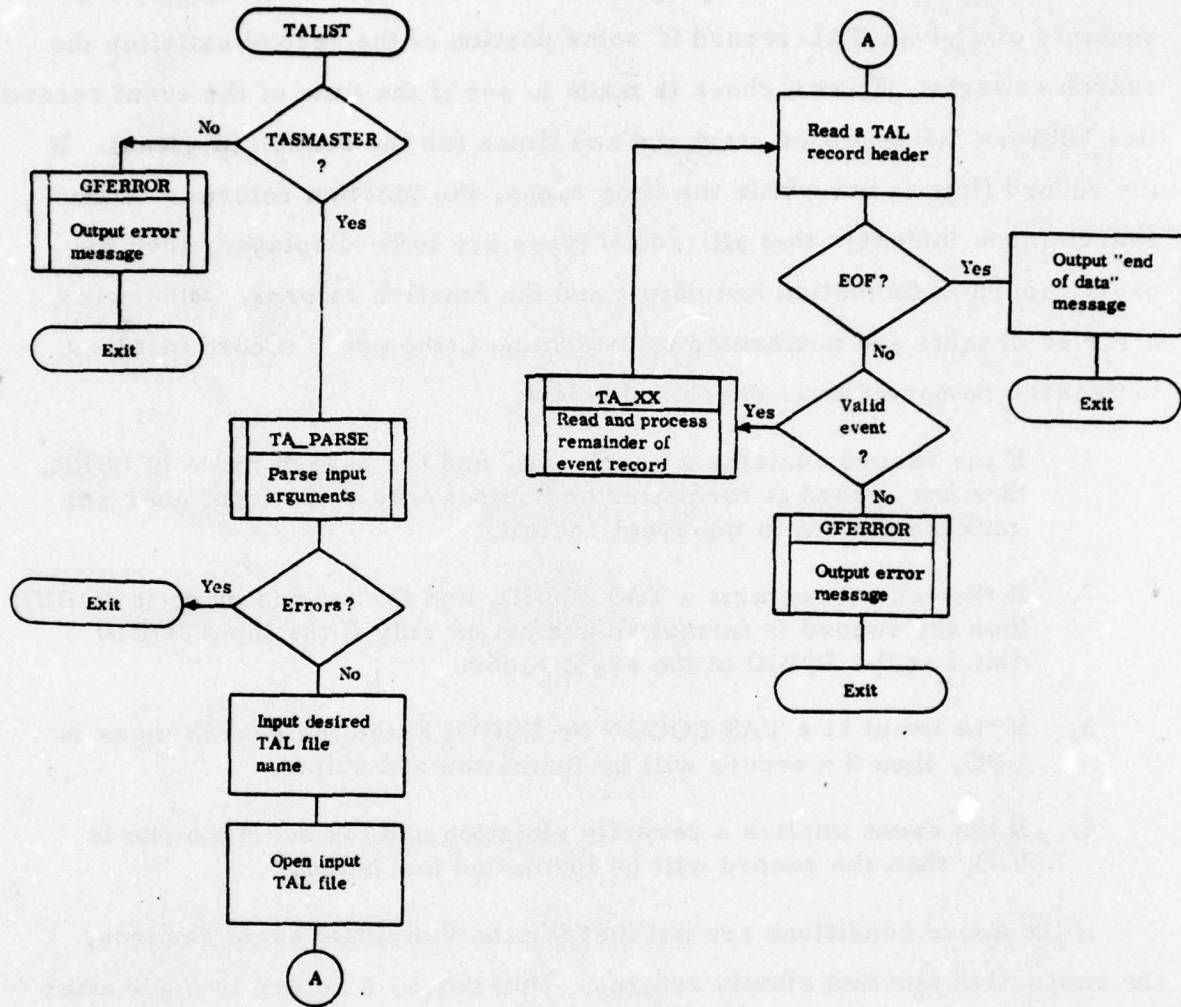


Figure 59. TALIST Process Data Flow (Sheet 1 of 18)

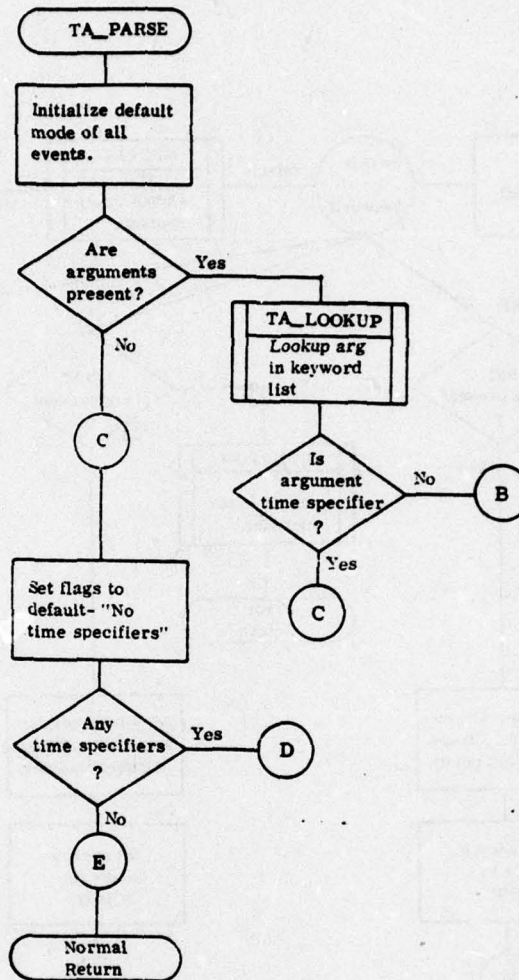


Figure 59. TALIST Process Data Flow (Sheet 2 of 18)

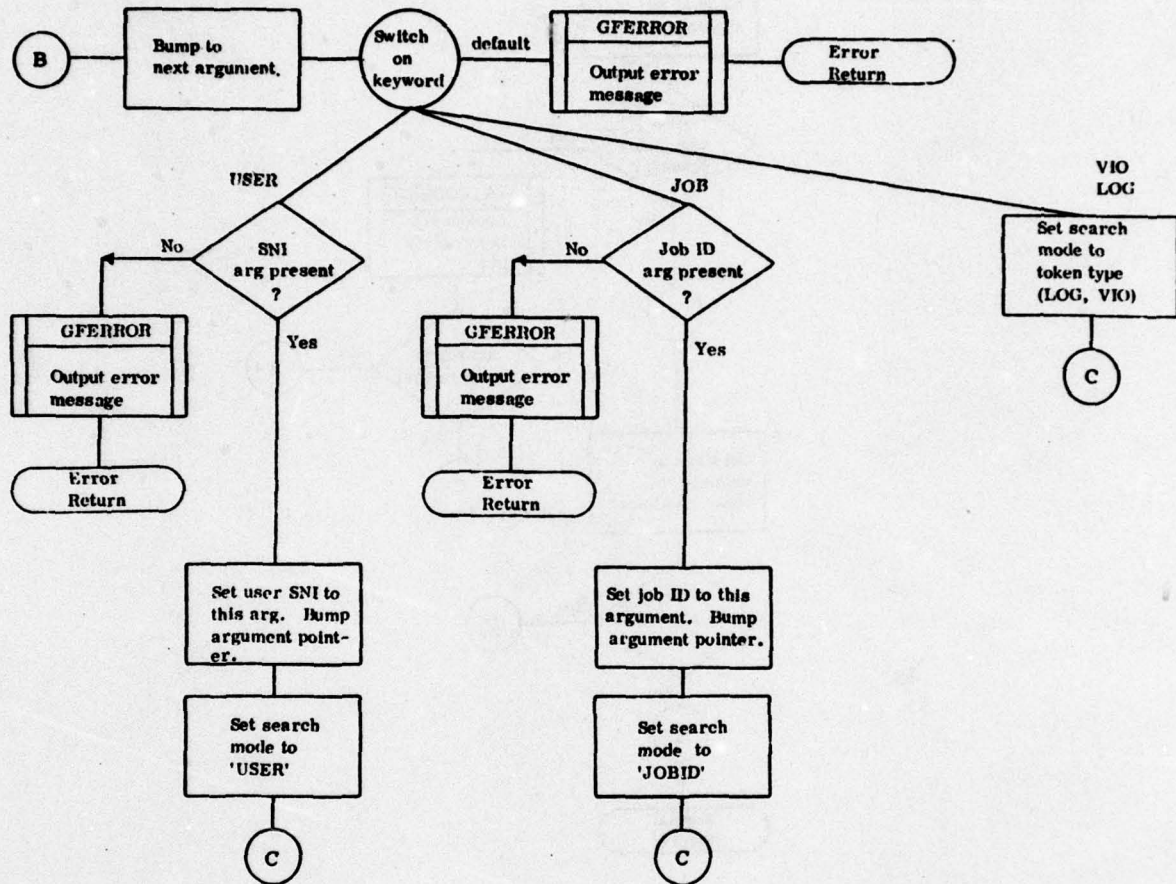


Figure 59. TALIST Process Data Flow (Sheet 3 of 18)

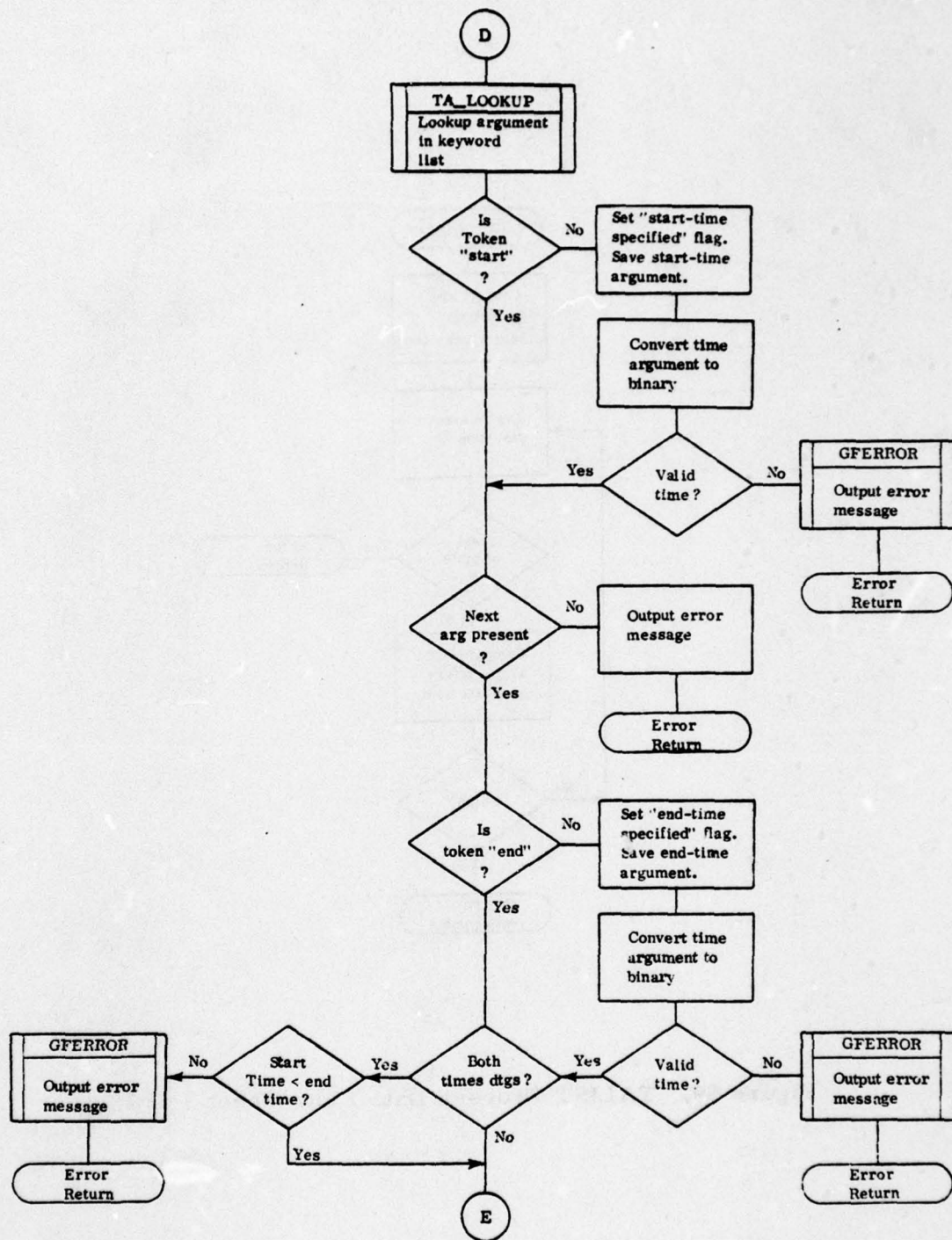


Figure 59. TALIST Process Data Flow (Sheet 4 of 18)

AD-A050 967

LOGICON INC SAN DIEGO CALIF

F/G 9/2

TERMINAL ACCESS SYSTEM (TAS) SYSTEM DESIGN SPECIFICATION. (U)

JAN 78 R M BARNHART, L R ERICKSON

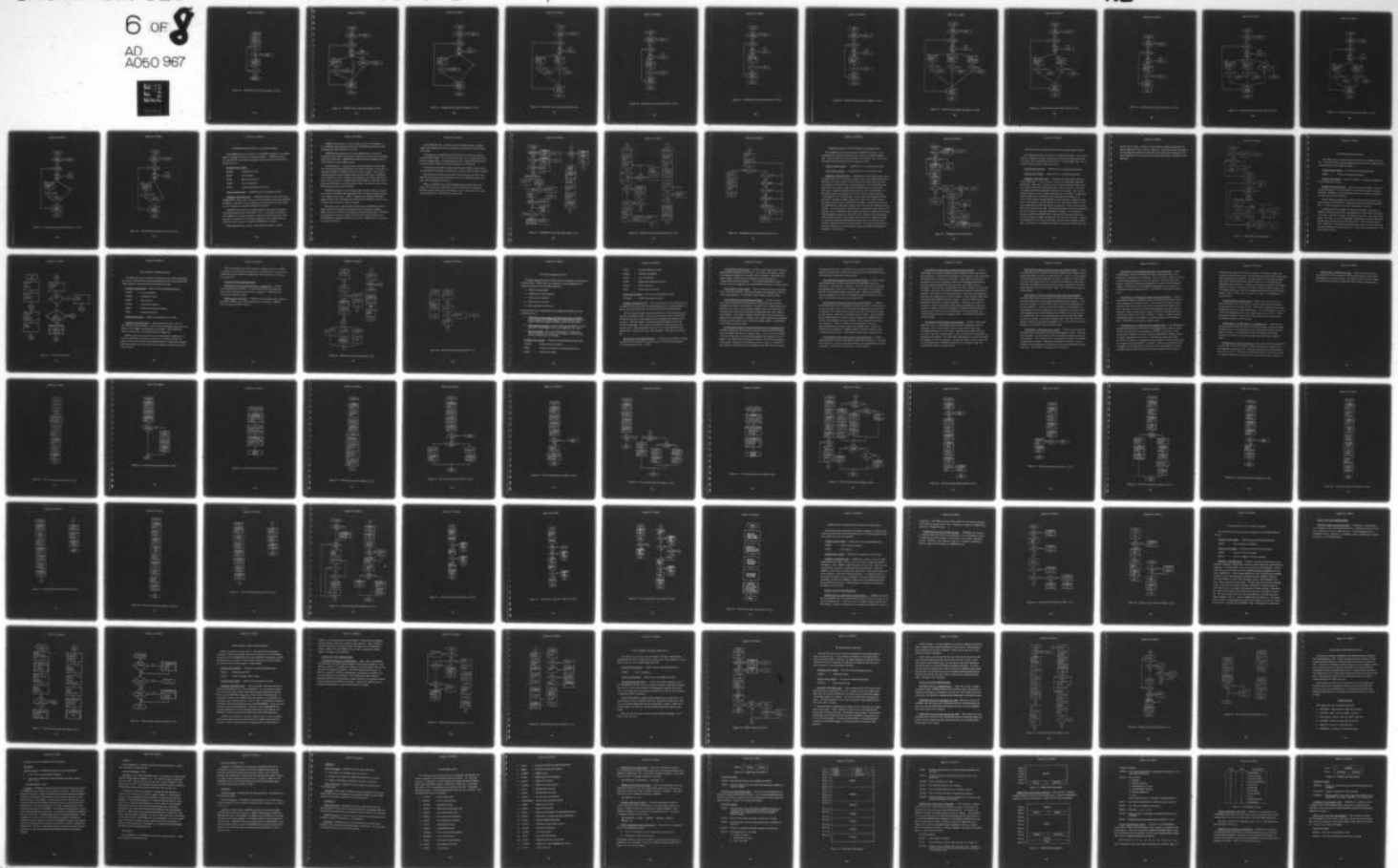
N00014-76-C-0899

UNCLASSIFIED

76-C-0899-6

NL

6 OF 8
AD
A050 967



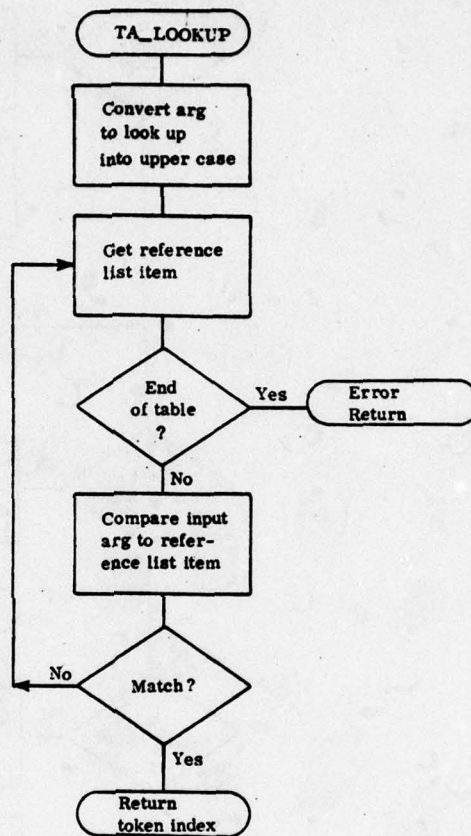


Figure 59. TALIST Process Data Flow (Sheet 5 of 18)

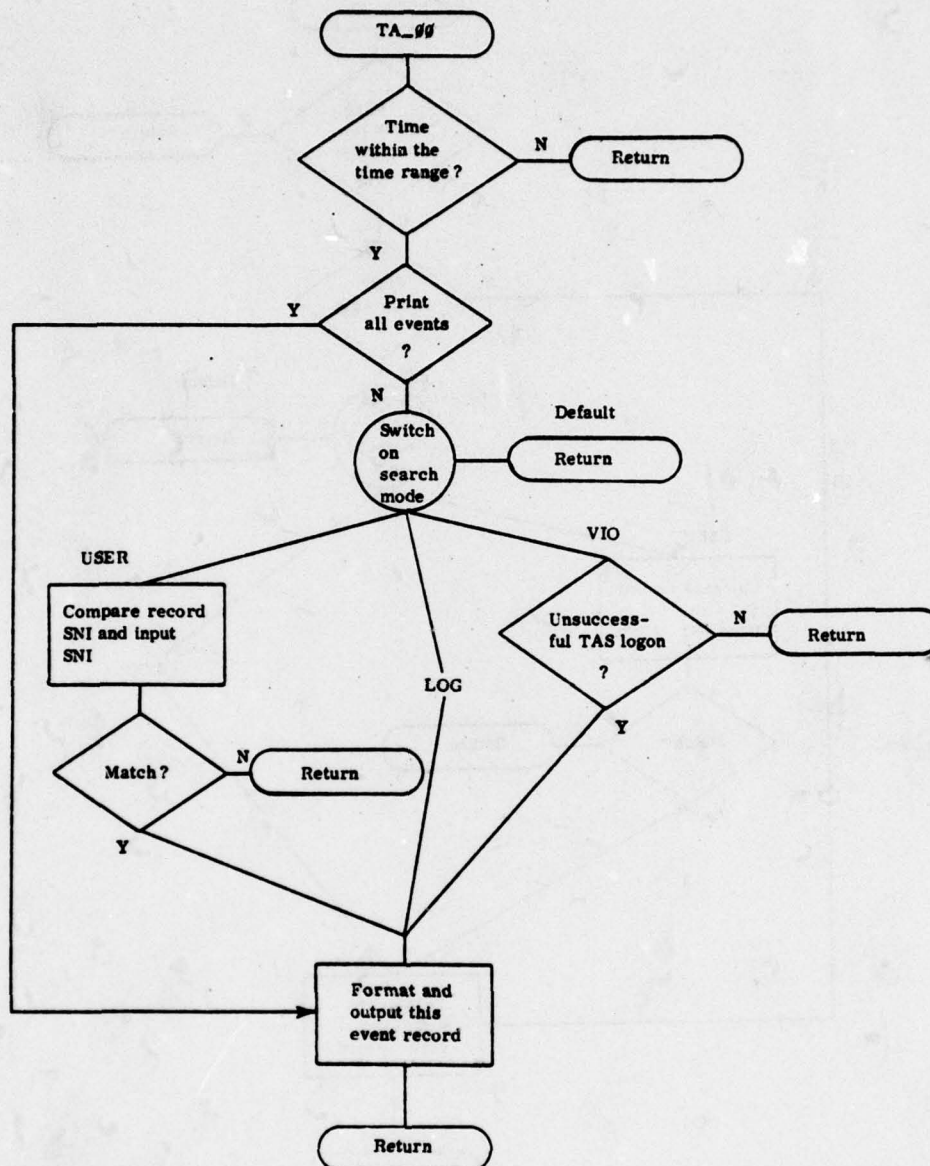


Figure 59. TALIST Process Data Flow (Sheet 6 of 18)

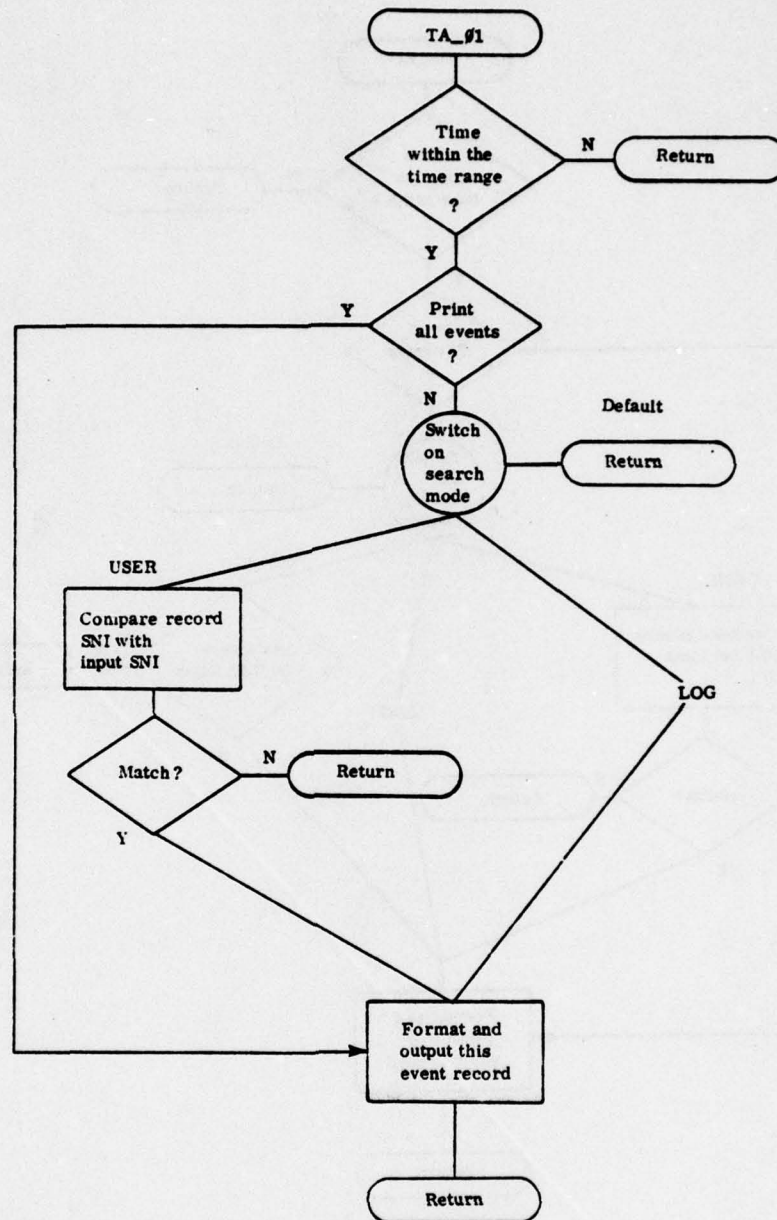


Figure 59. TALIST Process Data Flow (Sheet 7 of 18)

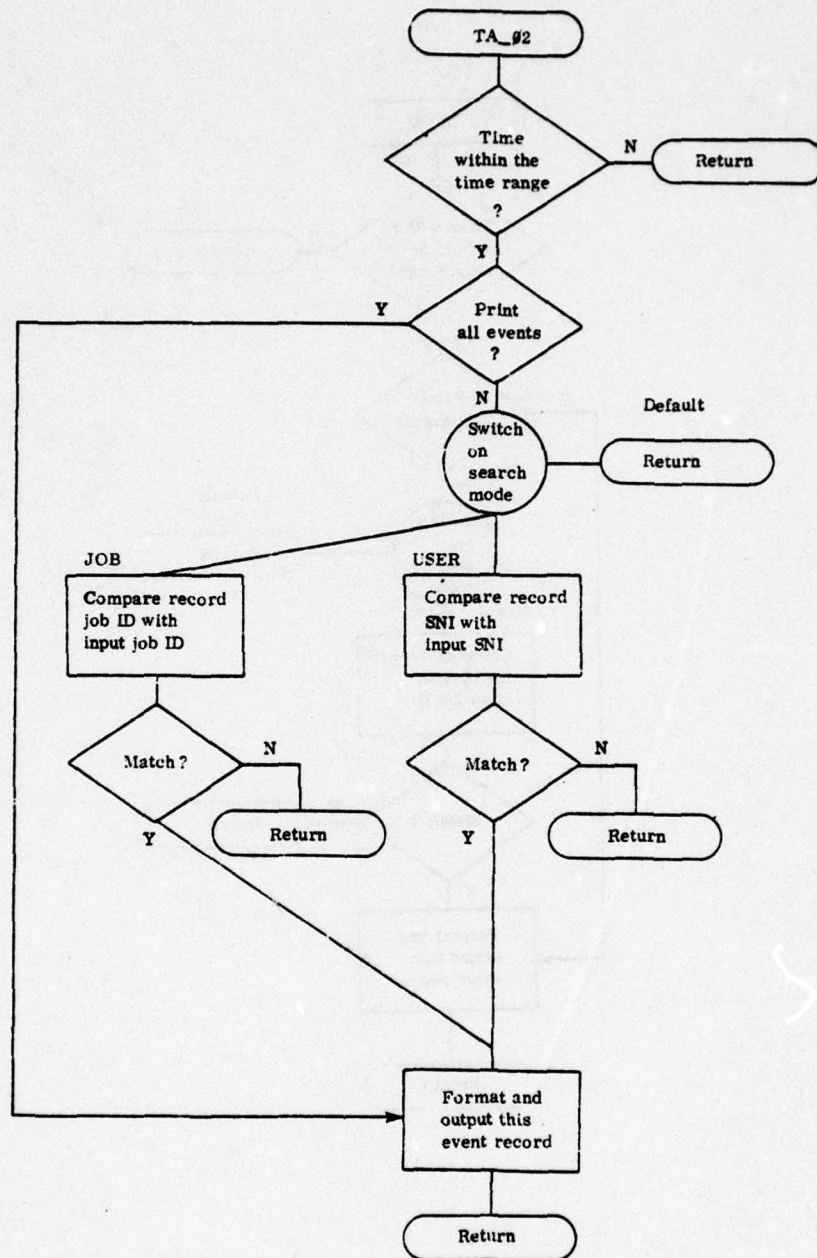


Figure 59. TALIST Process Data Flow (Sheet 8 of 18)

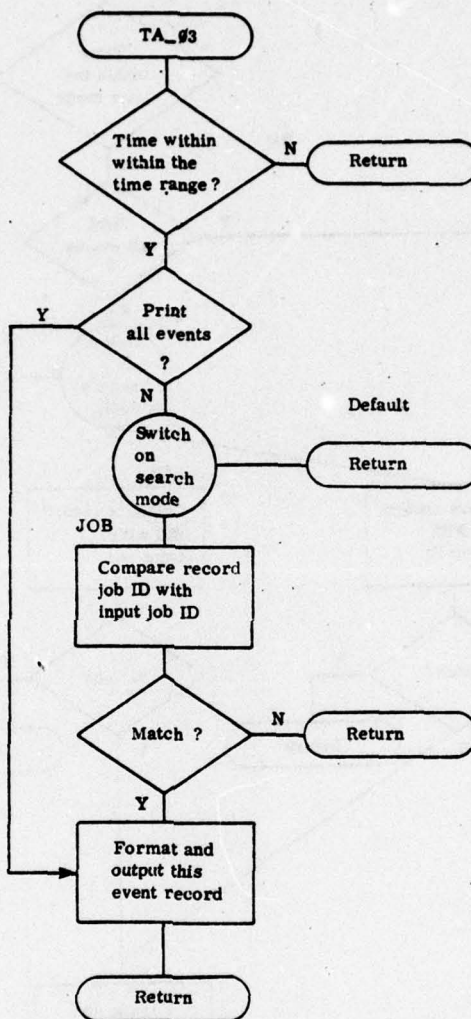


Figure 59. TALIST Process Data Flow (Sheet 9 of 18)

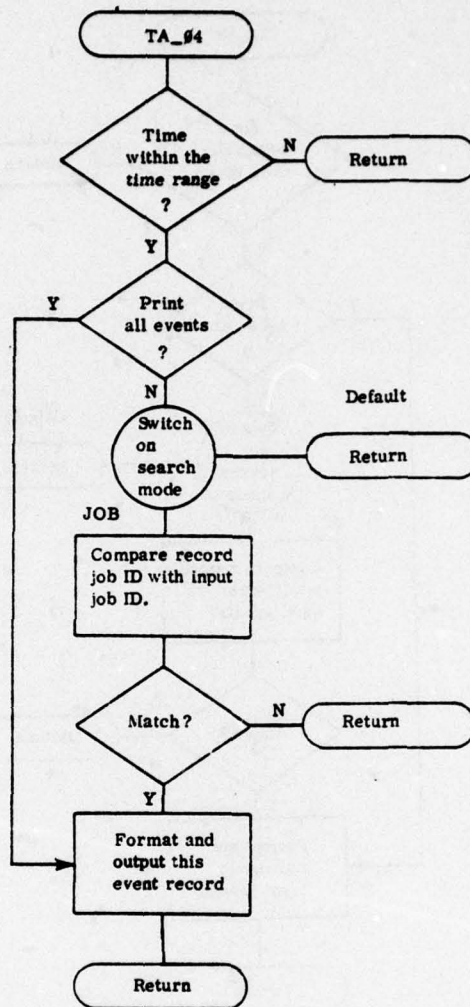


Figure 59. TALIST Process Data Flow (Sheet 10 of 18)

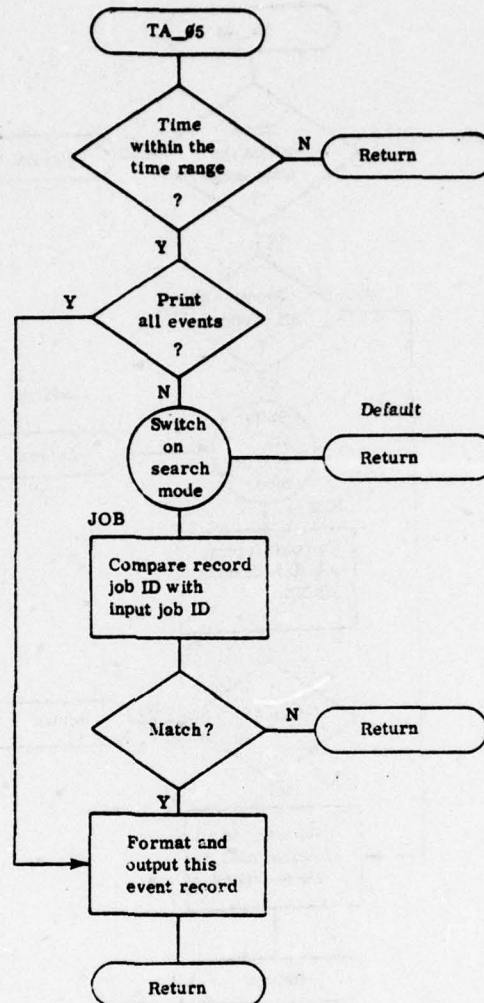


Figure 59. TALIST Process Data Flow (Sheet 11 of 18)

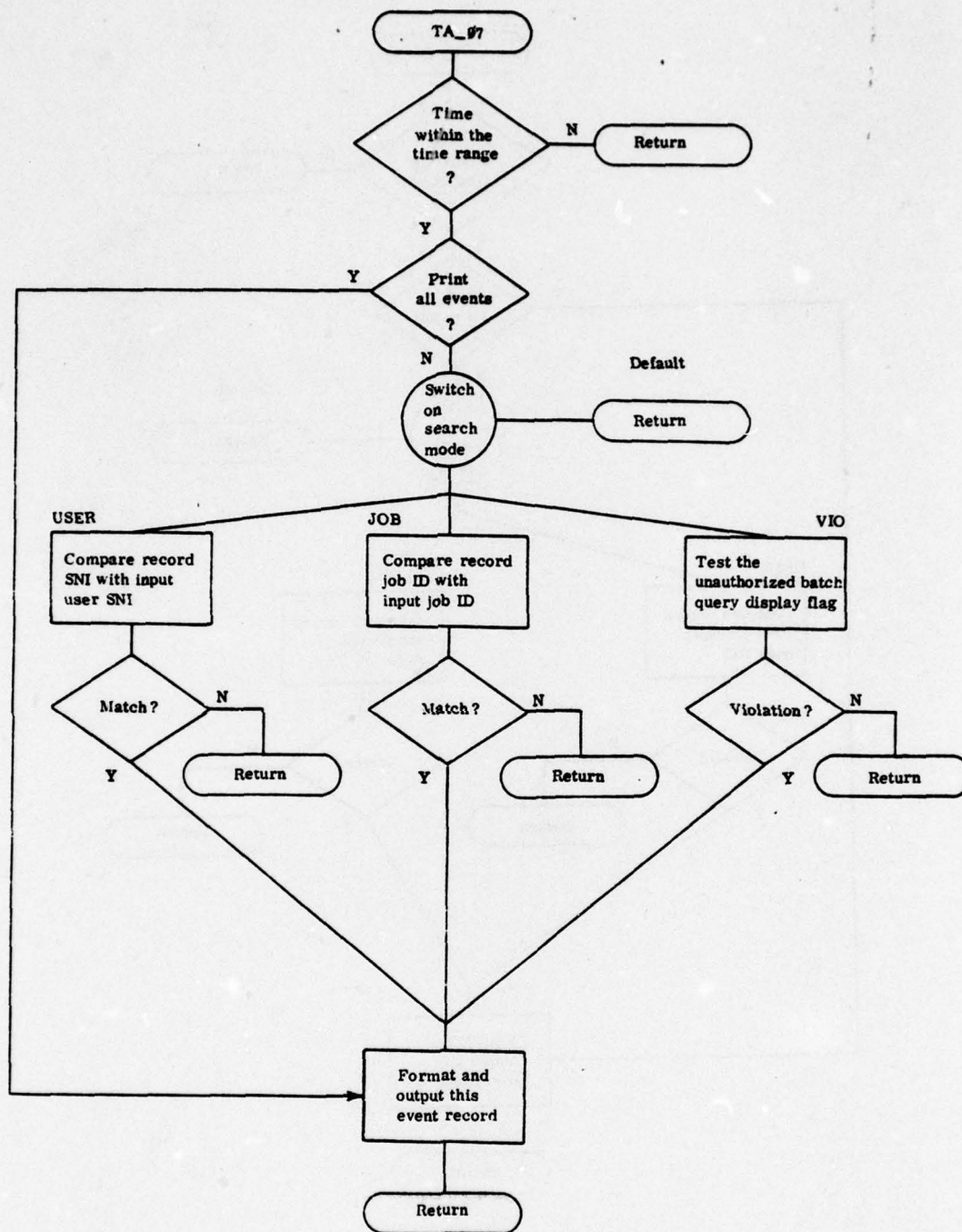


Figure 59. TALIST Process Data Flow (Sheet 12 of 18)

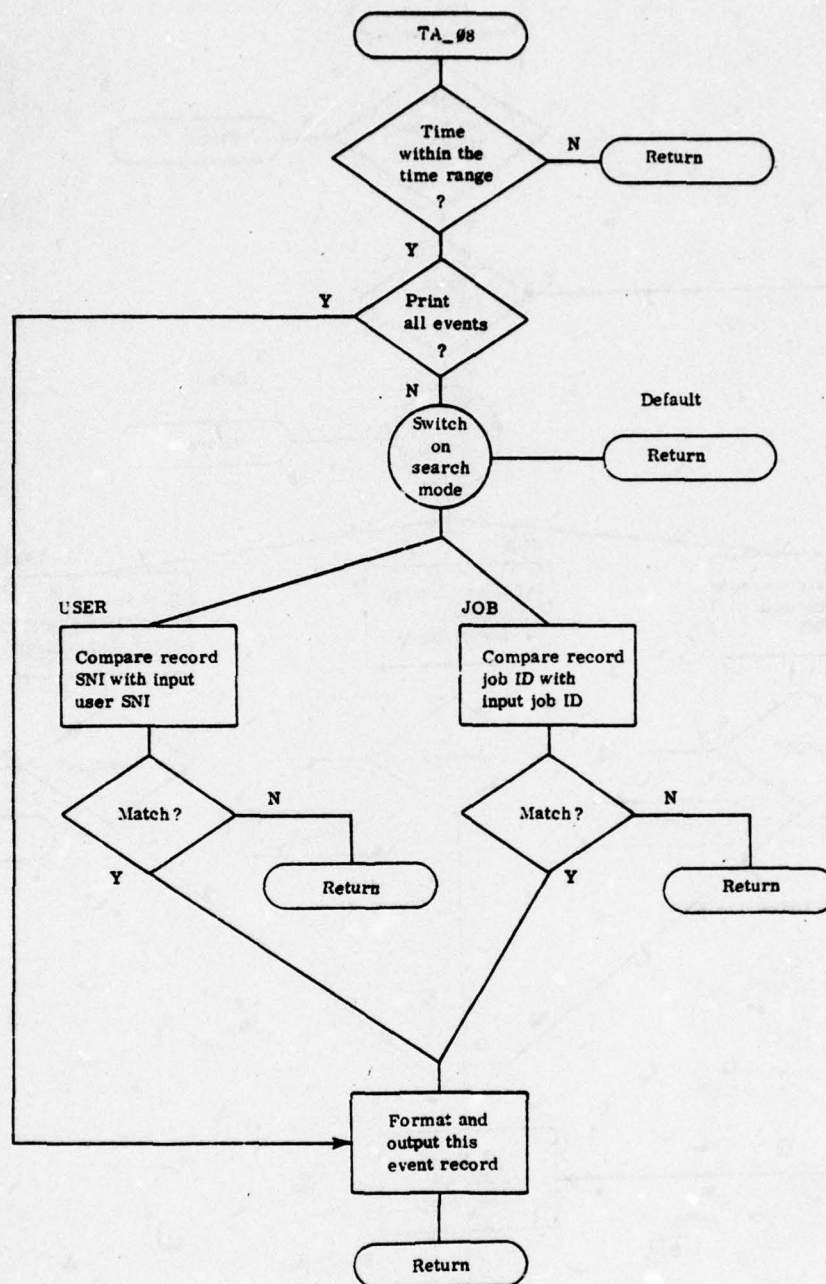


Figure 59. TALIST Process Data Flow (Sheet 13 of 18)

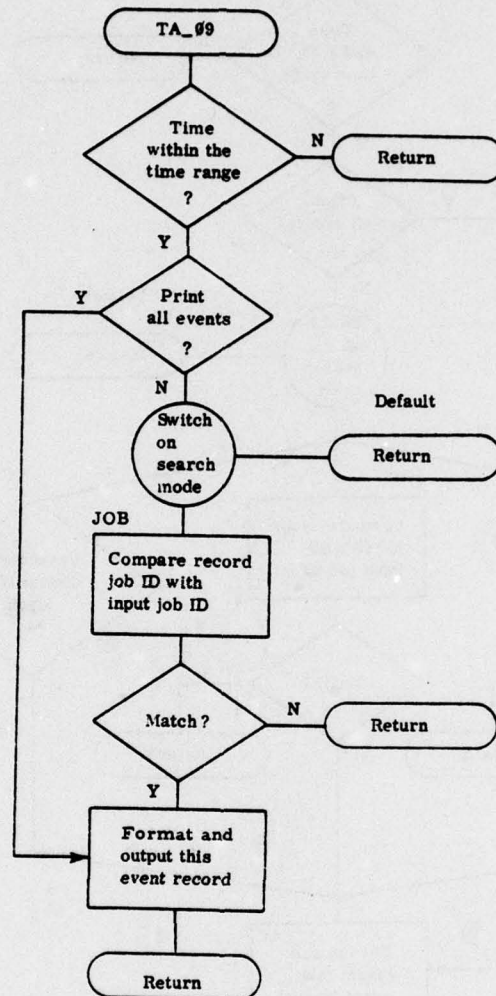


Figure 59. TALIST Process Data Flow (Sheet 14 of 18)

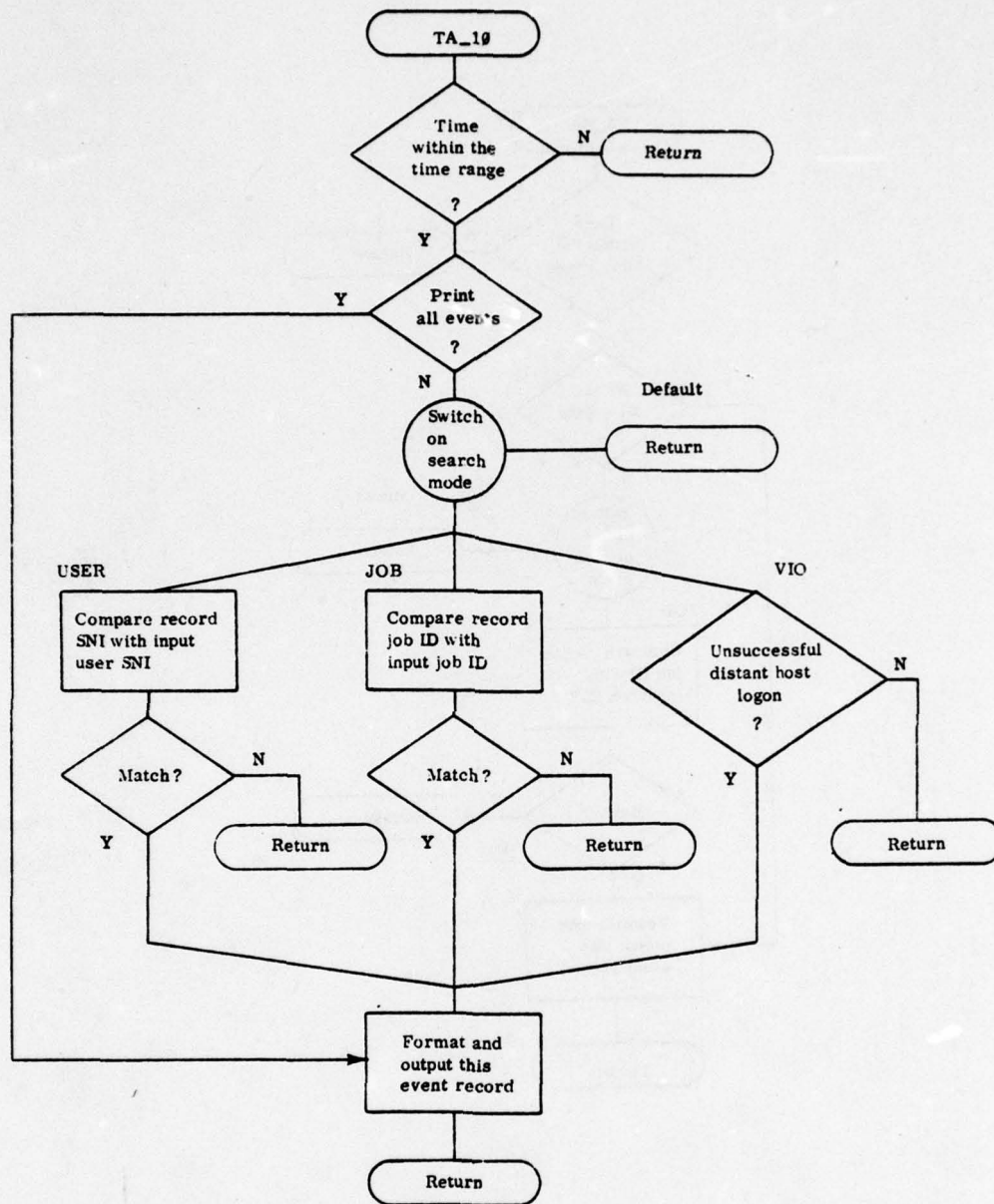


Figure 59. TALIST Process Data Flow (Sheet 15 of 18)

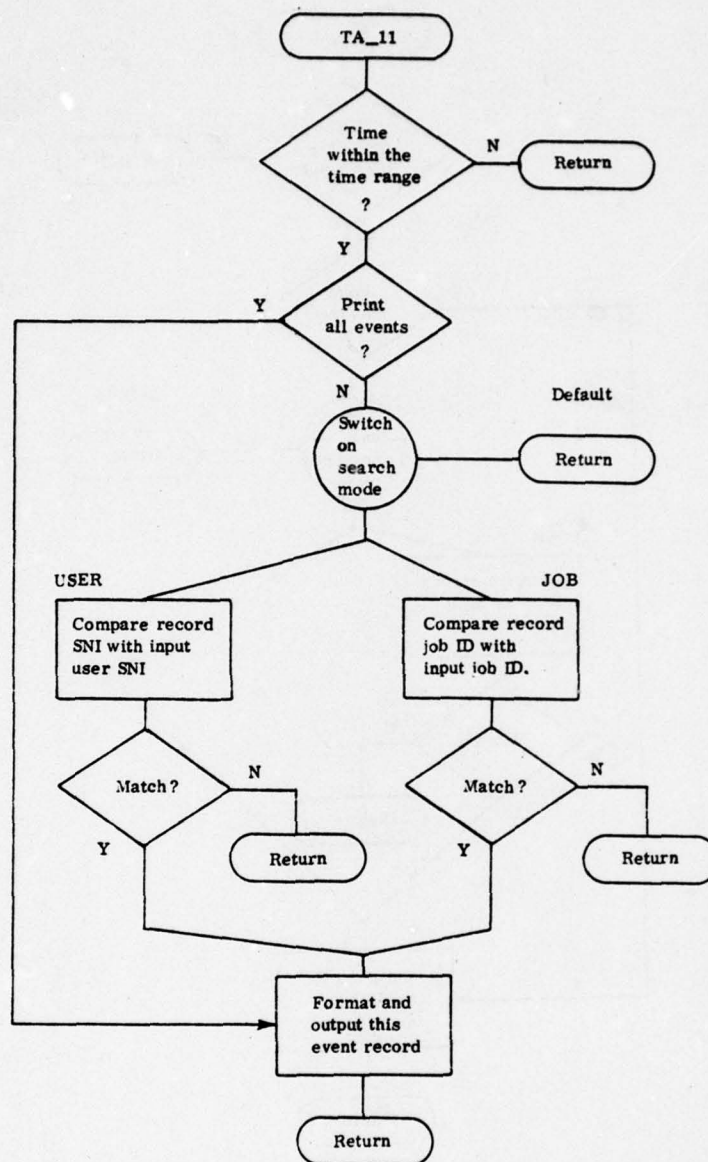


Figure 59. TALIST Process Data Flow (Sheet 16 of 18)

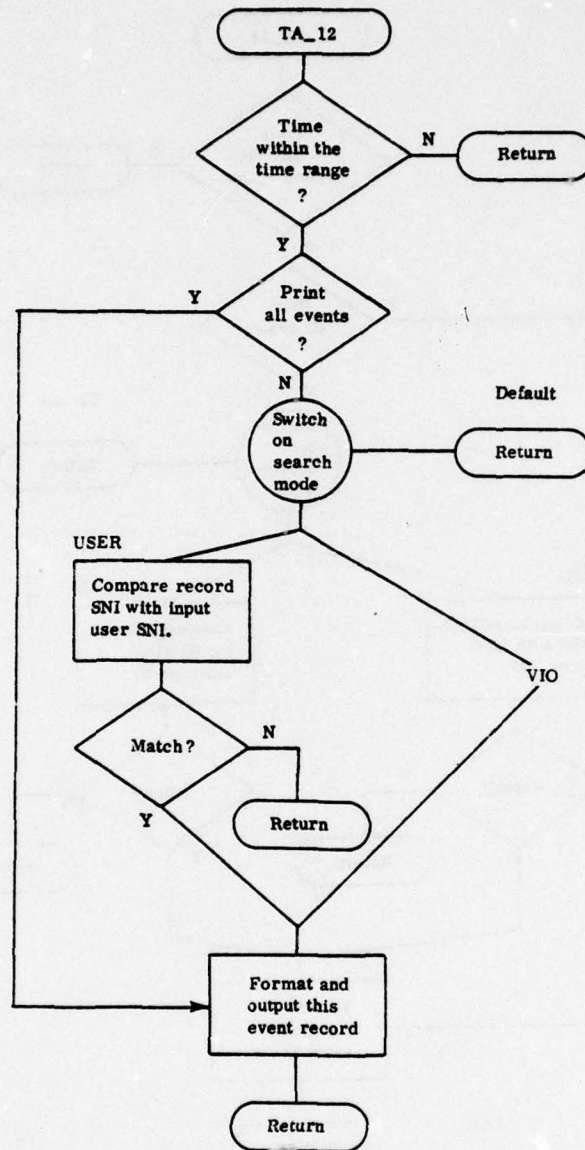


Figure 59. TALIST Process Data Flow (Sheet 17 of 18)

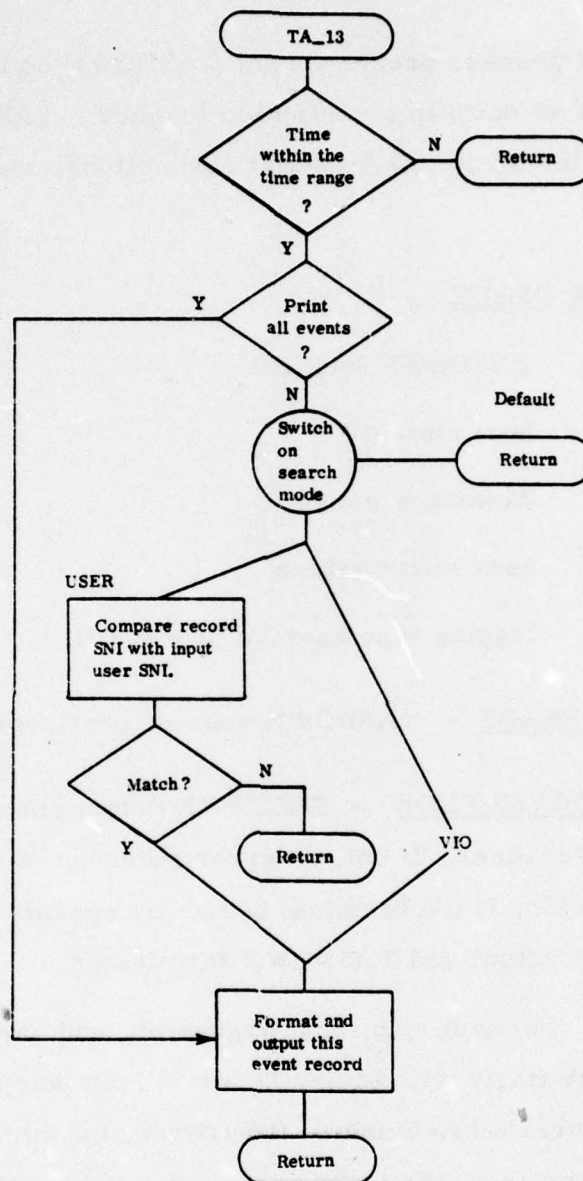


Figure 59. TALIST Process Data Flow (Sheet 18 of 18)

TASDOWN (Bring TAS Down in a Controlled Manner)

The TASDOWN process processes the TASDOWN command. It is responsible for bringing TAS down in a controlled manner. TASDOWN will log all users off TAS, terminate all terminal connections, and kill off all background processes.

GLOBAL DATA USAGE

GDOWN	-	TASDOWN lock file
GHOST	-	host statuses
GPIDS	-	daemon process IDs
GUIDES	-	user descriptions
LTPID	-	logging tape daemon process ID

LOCAL DATA USAGE - TASDOWN uses no pertinent local data.

GENERAL PROCESS FLOW - TASDOWN determines if the user is the TASMMASTER or super-user. If not, an error message is output and TASDOWN terminates. Otherwise, if the terminal is not the operator's console (tty8), an error message is output and TASDOWN terminates.

TASDOWN then checks for an input argument, and saves it if present. If the input argument starts with either '+' or '-', the number of minutes until shutdown is converted to binary. Otherwise, the time of day to cease operation is calculated from the input argument. If the resultant time is invalid, an error message is output and TASDOWN terminates.

If the shutdown time is valid, a TAS shutdown message is output.

TASDOWN determines if it is more than one hour until shutdown. If so, TASDOWN sleeps until one hour before the shutdown time and sets the remaining time until shutdown to one hour.

TASDOWN then determines if it is the shutdown time. If not, the broadcast process SDOWN is executed to inform users of the time remaining until TAS is shut down. TASDOWN then sleeps half the remaining time and repeats this cycle until shutdown time.

At shutdown time, a check is made to ensure that TAS is in the single user mode (as indicated by 00773030 in the console switches). If it is not, a message is output and TASDOWN sleeps for three seconds. This cycle is repeated until the console switches have been set to 00773030. TASDOWN next creates the lock file, GDOWN, which indicates that TAS is down and no more users may log on. TASDOWN then locks GHOST via GFLOCK, reads item zero (TAS status), sets TAS status to down, rewrites item zero, closes GHOST and unlocks it via GFUNLOCK.

TASDOWN then gives each remaining TAS user three chances to log off. For each user in GUDS still logged onto TAS, a warning message is output and a logged-on flag is set. If after a complete pass through GUDS the logged-on flag is set, indicating users still on TAS, SDOWN is called to warn all users to log off and the cycle is repeated.

Otherwise, when a pass through all TAS users in GUDS shows no one logged onto TAS or when the third pass has been completed, a hangup signal is sent to the INIT process killing all terminal connections. TASDOWN then sleeps for 15 seconds, raises its priority and saves its process ID. TASDOWN then executes the SRVC process with a "BREAK ALL" and sleeps for another 10 seconds.

Next TASDOWN opens, reads and closes LTPID and sends a hangup signal to the logging tape daemon and sleeps for another 10 seconds waiting for it to die.

Following this, TASDOWN locks (via GFLOCK) and opens the daemon process ID file GPIDS. For each PID entry read, the corresponding process is sent a hangup signal, terminating that process. When all TAS processes have been killed, GPIDS is closed and unlocked via GFUNLOCK.

The TAS network traffic summary report process TSUM is then called via a fork/execute sequence, and generates a report on the operator's console.

The UNIX process table is then read and for each process listed in the table which is not owned by daemon (ID 1) or is not TASDOWN itself, a signal is sent terminating that process.

Finally, TASDOWN waits until all background processes have died, sleeps for 60 seconds, performs three SYNC system calls to update the UNIX file system super block, then outputs a final "TAS is shut down" message and terminates. See figure 60 for data flow.

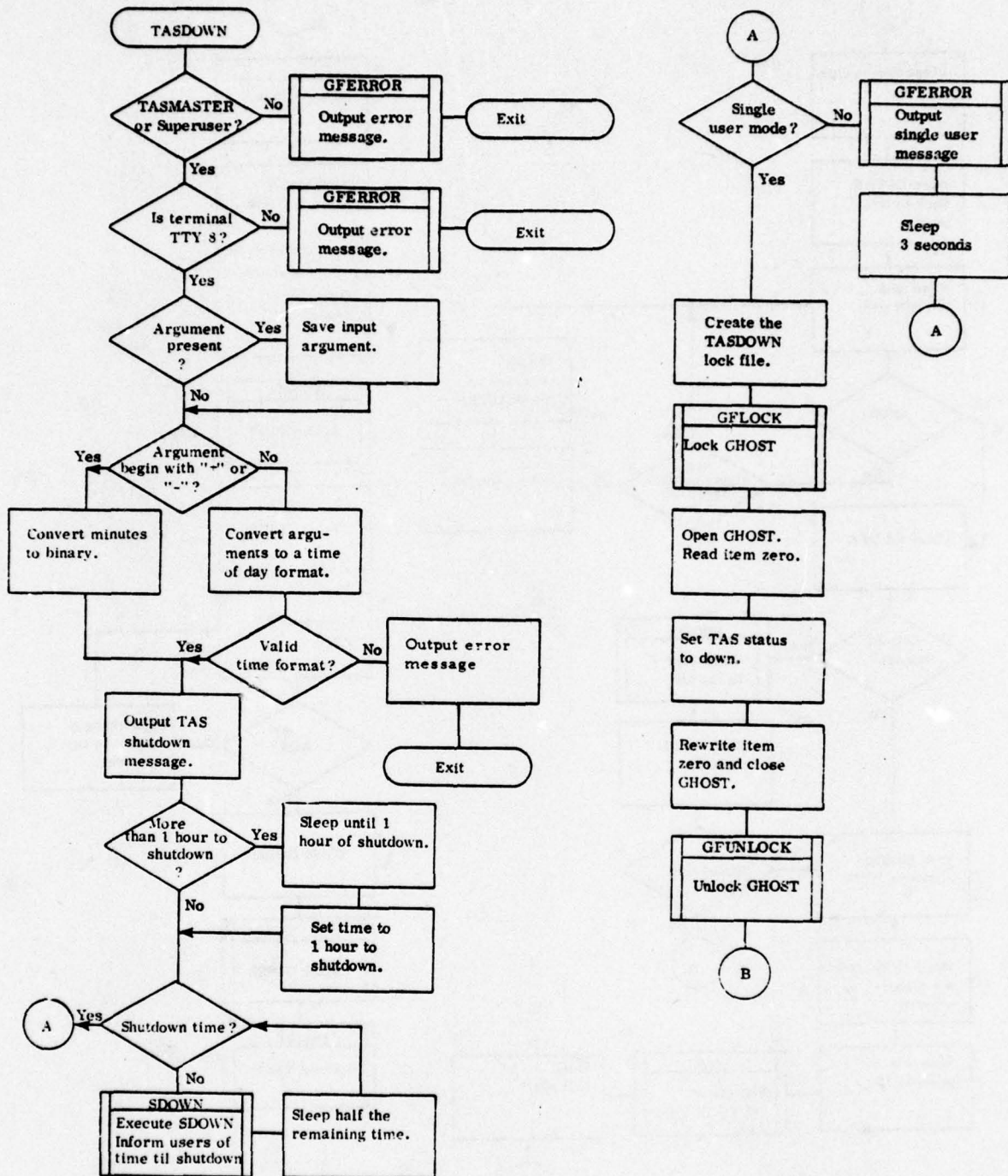


Figure 60. TASDOWN Process Data Flow (Sheet 1 of 3)

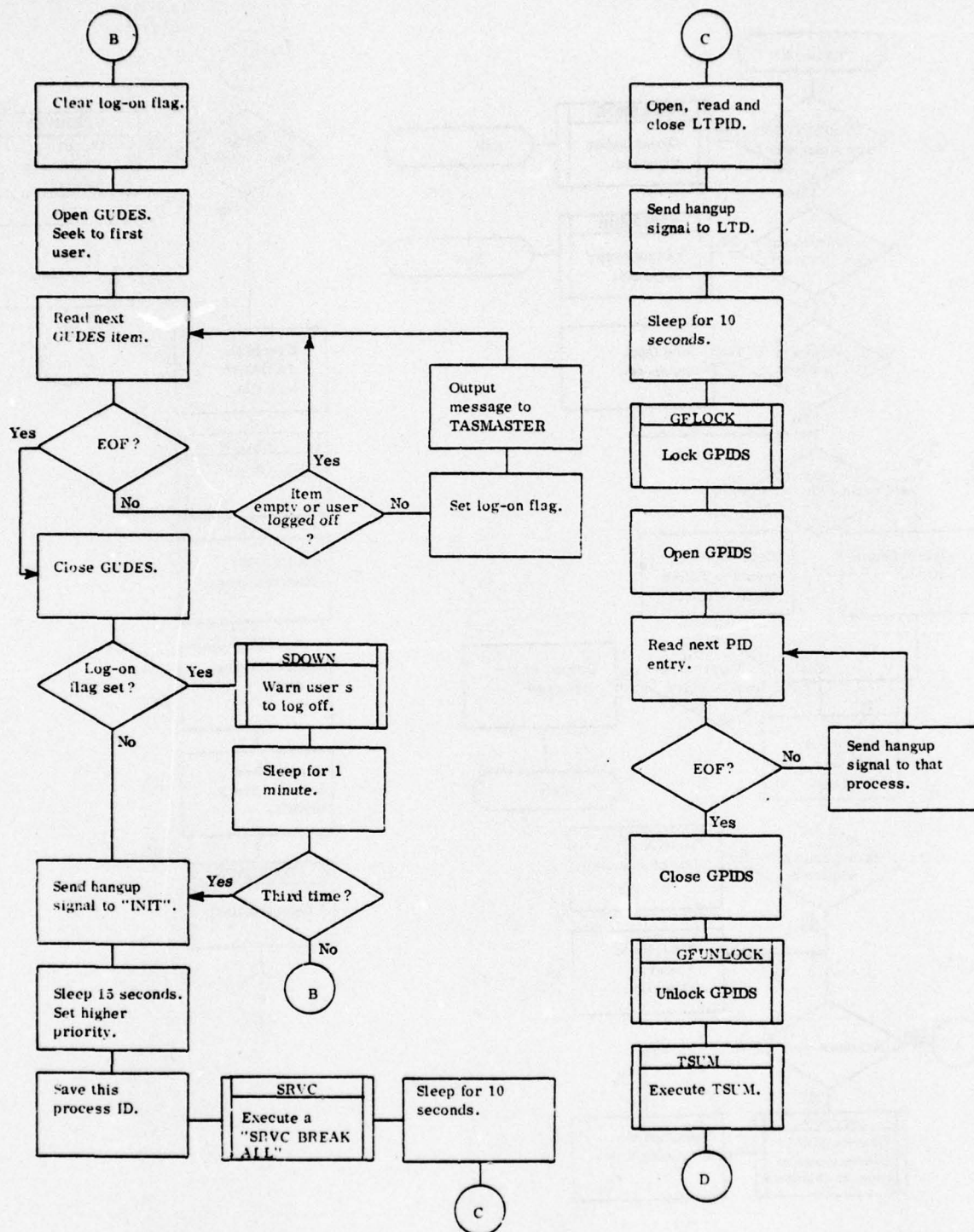


Figure 60. TASDOWN Process Data Flow (Sheet 2 of 3)

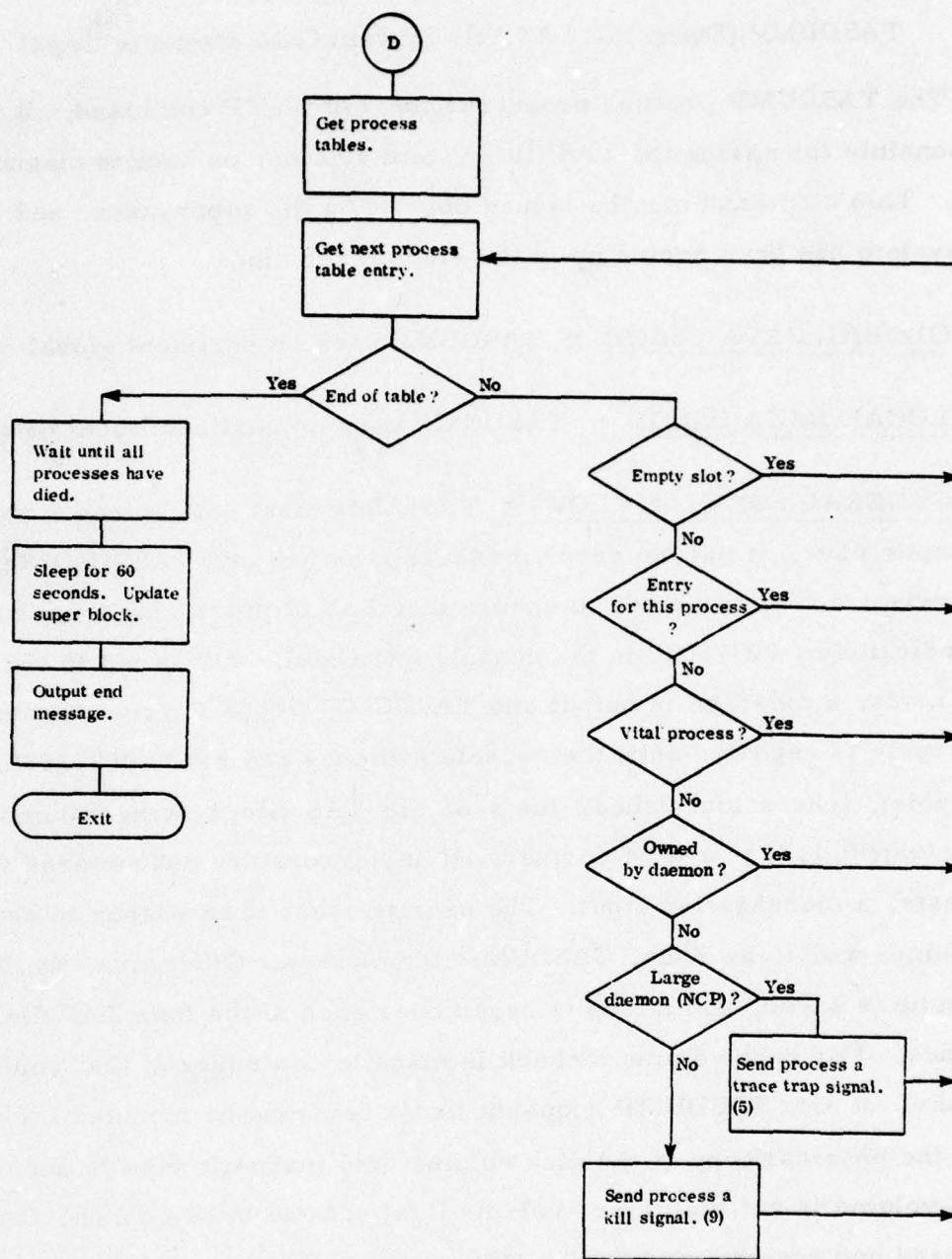


Figure 60. TASDOWN Process Data Flow (Sheet 3 of 3)

TASDUMP (Dump the TAS File System Onto Magnetic Tape)

The TASDUMP process processes the TASDUMP command. It is responsible for saving the TAS file system volumes on backup magnetic tape. This command may be issued only 1) by the super-user, and 2) after the system has been booted up in the single user mode.

GLOBAL DATA USAGE - TASDUMP uses no pertinent global data.

LOCAL DATA USAGE - TASDUMP uses no pertinent local data.

GENERAL PROCESS FLOW - TASDUMP first determines if the user is the super-user. If not, an error message is output and TASDUMP terminates. Otherwise, a check is made to ensure that TAS is in the single user mode (as indicated by 00773030 in the console switches). If it is not in the single user mode, a message is output and TASDUMP sleeps for five seconds. This cycle is repeated until the console switches are set to 00773030. At that point, icheck and dcheck tests of the TAS file system volumes "/dev/rhp00,1,2,3" are performed. If any errors are encountered during the tests, a message is output. The user is asked if he wishes to abort the dump, and if he does, TASDUMP terminates. Otherwise, the time of the dump is saved, and a loop is begun over each of the four TAS file system volumes. For each volume a check is made to determine if that volume is the last one. If not, TASDUMP requests that a new tape be mounted for the dump. Then the physical copy of the disk volume onto magnetic tape is performed. If the volume is not the "/usr" volume (last volume to be copied), the tape is closed and rewound and TASDUMP instructs the user to label the tape. Finally, if the last volume has not been copied, the loop is repeated for the next one; otherwise, an end message is output and TASDUMP terminates. See figure 61 for data flow.

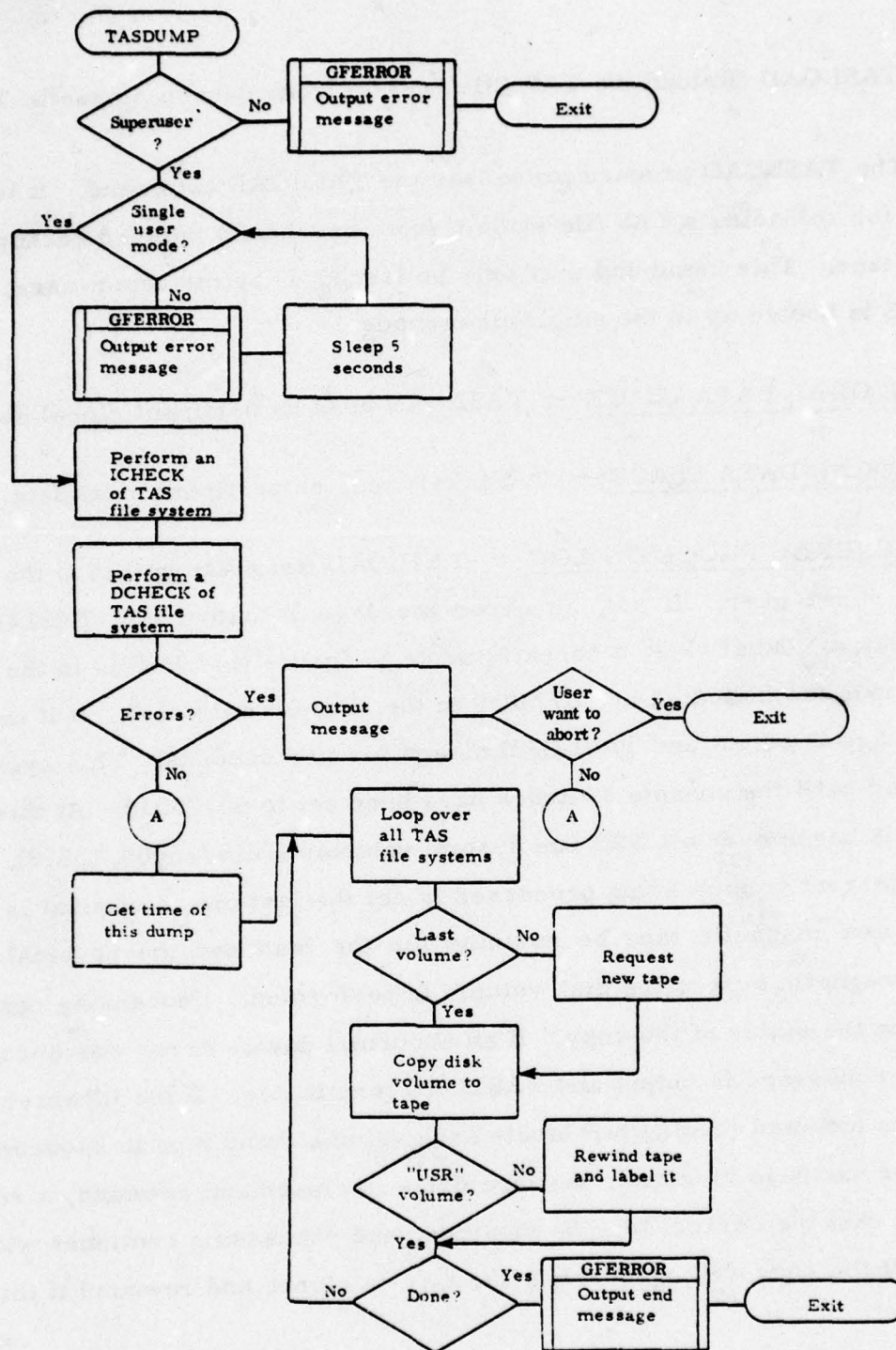


Figure 61. TSDUMP Process Data Flow

TASLOAD (Reload the TAS File System from Backup Magnetic Tape)

The TASLOAD process processes the TASLOAD command. It is responsible for reloading a TAS file system from previously dumped backup magnetic tape. This command may only be issued 1) by the super-user, and 2) if TAS is booted up in the single-user mode.

GLOBAL DATA USAGE - TASLOAD uses no pertinent global data.

LOCAL DATA USAGE - TASLOAD uses no pertinent local data.

GENERAL PROCESS FLOW - TASLOAD first determines if the user is the super-user. If not, an error message is output and TASLOAD terminates. Otherwise, a check is made to ensure that TAS is in the single user mode (as indicated by 00773030 in the console switches). If it is not, a message is output and TASLOAD sleeps for five seconds. This cycle is repeated until the console switches have been set to 00773030. At this point a loop is begun over all TAS file system volumes (/dev/rhp00, 1, 2, 3). If the current volume being processed is not the last one, a request is made that a new magnetic tape be mounted for the load and the physical copy of the magnetic tape to the disk volume is performed. Processing continues based on the status of the copy. If an abnormal device error was encountered, an error message is output and TASLOAD terminates. If the incorrect tape was mounted (TASDUMP labels each volume dump tape in sequence), an error message is output, the tape drive is closed and rewound, a request is made that the correct tape be mounted, and processing continues with the copy. If the copy was successful, the tape is closed and rewound if this was

not the "/usr" volume. Finally, if more volumes remain to be loaded, the loop is repeated for those volumes; otherwise, TASLOAD outputs instructions for restoring TAS multi-user operation, then sleeps forever waiting for the user to halt the CPU and execute the specified instructions. See figure 62 for data flow.

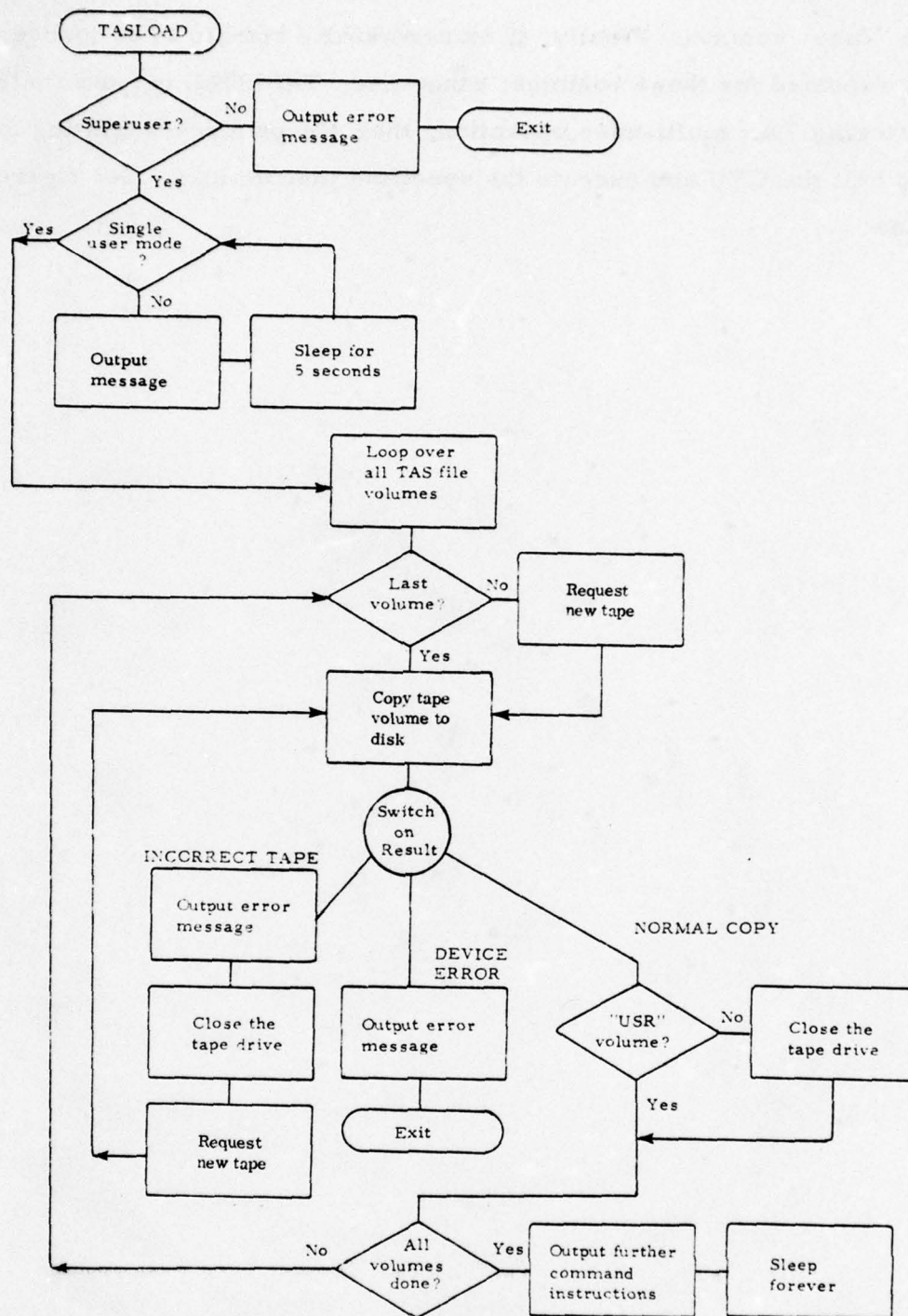


Figure 62. TASLOAD Process Data Flow

TD (Network Time-out Daemon)

The TD process is a daemon which is activated via the UNIX "/etc/rc" file. TD checks batch messages against a time-out interval in order to cause retransmission of unreceipted messages.

GLOBAL DATA USAGE - TD uses the following global data:

GIMSG - INI batch message descriptions

LOCAL DATA USAGE - TD uses TDTIMEO, a variable containing the time-out interval in seconds.

GENERAL PROCESS FLOW - TD first ignores all signals. If there is an input parameter, TD sets TDTIMEO to the specified time-out interval; otherwise, TDTIMEO is set to the default value, four hundred seconds (as specified in the COINS I protocol).

TD then begins an infinite loop, wherein TD sleeps for one-half of the time-out interval ($TDTIMEO/2$), awakens and checks all Intelligent Network Interface (INI) messages for time-out, and returns to its sleep.

In checking the INI messages, TD first opens GIMSG. Each item in GIMSG is read until end-of-file, at which point TD closes GIMSG and returns to its sleep. If an item is empty, TD reads the next item. If an item is not empty, TD checks to see if the message's transmission time was longer ago than the time-out interval from the current time. If not, the next item is read. Otherwise, TD performs a fork/execute sequence to call the INI process with a function of message timed out. TD then reads the next item. See figure 63 for data flow.

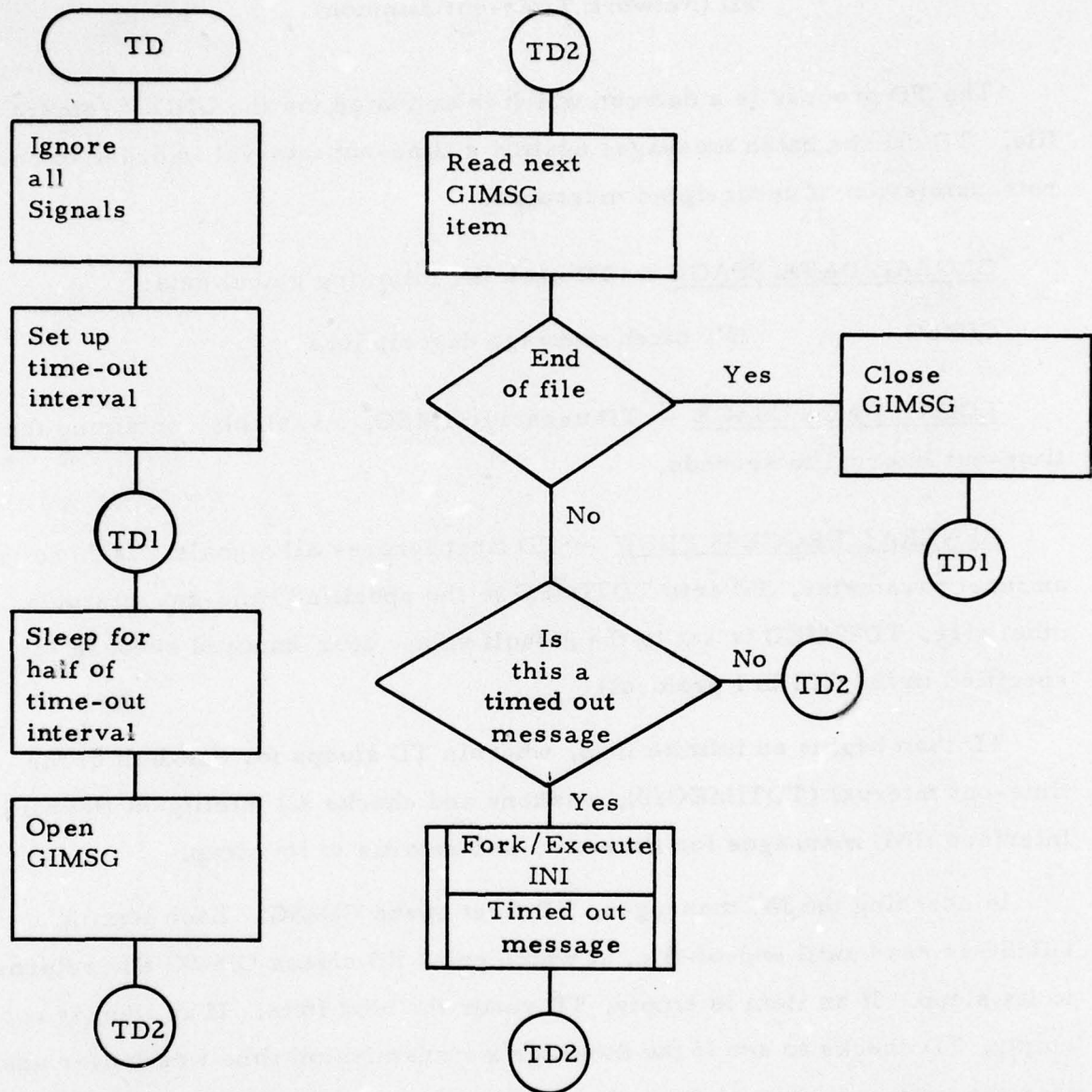


Figure 63. TD Process Data Flow

TINIT (Perform TAS Initialization)

The TAS system process TINIT is activated from the UNIX initialization file "/etc/rc" when TAS is brought up and performs all system initialization required before TAS can function in the multi-user mode.

GLOBAL DATA USAGE - TINIT uses the following global data:

GUIDES	-	user descriptions
GDOWN	-	TASDOWN lock file
GHOST	-	host statuses
GTERM	-	terminal descriptions
GIMSG	-	INI batch message descriptions
GPIDS	-	daemon process IDs

LOCAL DATA USAGE - TINIT uses no pertinent local data.

GENERAL PROCESS FLOW - When activated from the "/etc/rc" file, TINIT first opens GUIDES. Each user item in GUIDES is read, the user's status is set to logged off and that item is rewritten. Then TITERM is called to set the status of all TAS terminals to logged off.

GHOST is then opened and read and each host status is initialized. If the host is TAS itself, the status is set to up. If the host is batch, the status is set to break and the write daemon PID file for that host is unlinked. If the host is interactive, the status is set to unknown, and the last up/down time is cleared.

When all TAS hosts have been processed, GHOST is rewritten, TIMSG is called to clear INI's batch message jobs, the GPIDS file is recreated, the TASDOWN lock file GDOWN is unlinked and TINIT terminates. See figure 64 for data flow.

MAJOR FUNCTION DESCRIPTIONS

TITERM (Set all TAS Terminal Statuses to Logged Off) - TITERM first opens and reads GTERM . For each terminal item in GTERM the status of the referenced terminal is set to logged off. GTERM is then rewritten and TITERM returns.

TIMSG (Clear all INI Jobs) - TIMSG first opens GIMSG and for each INI job listed in GIMSG, the corresponding data file is unlinked. GIMSG is then recreated and TIMSG returns.

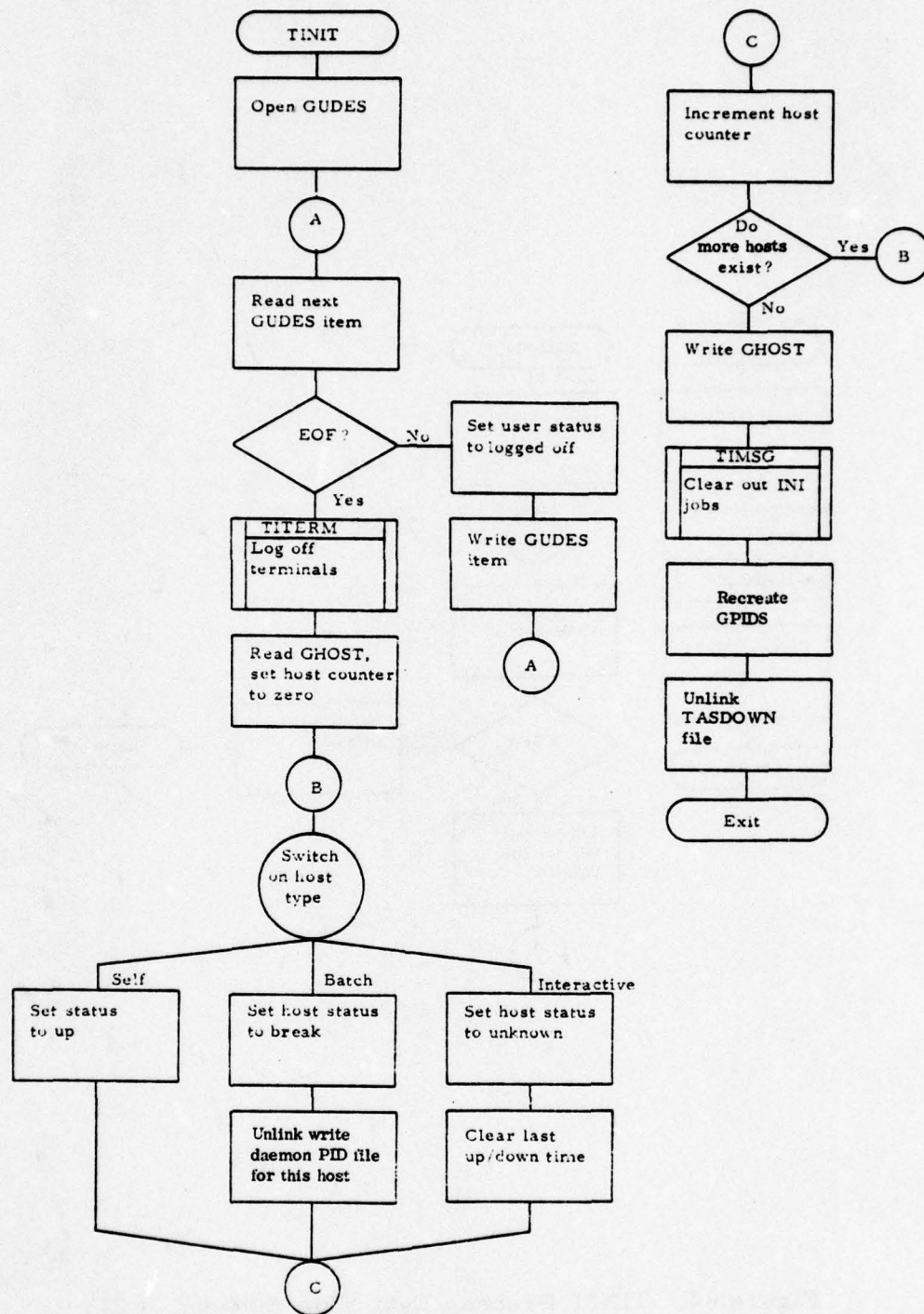


Figure 64. TINIT Process Data Flow (Sheet 1 of 2)

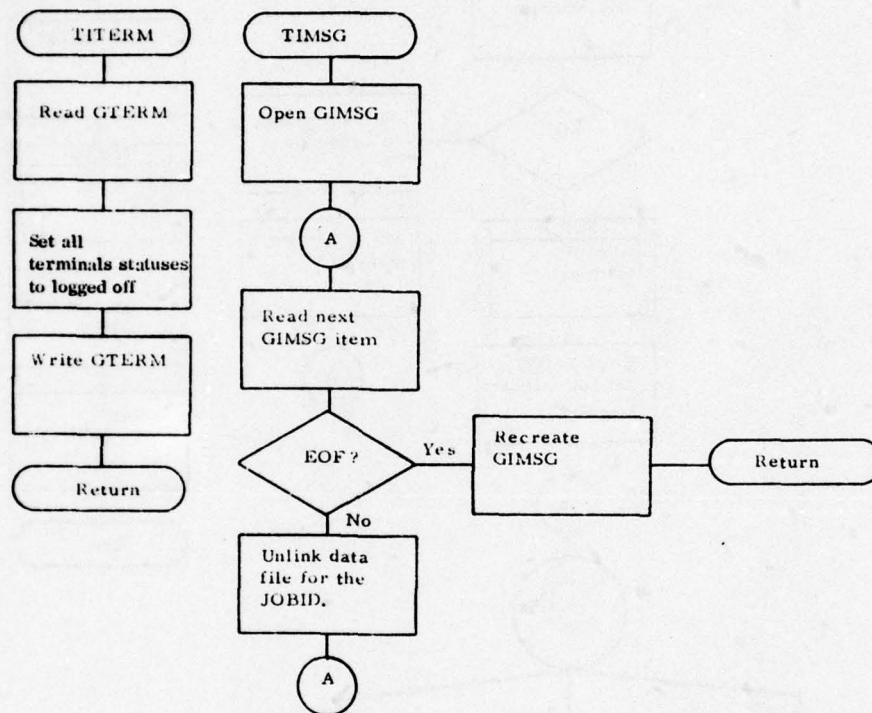


Figure 64. TINIT Process Data Flow (Sheet 2 of 2)

TLP (TAS Logging Processes)

The TAS Logging Processes (TLP) consist of fourteen event processors (named TLP00 - TLP13) which are activated to record significant TAS events which fall into the following categories:

1. TAS user logon/logoff
2. COINS network interrogations
3. TAS security violations
4. jobs lost by the network
5. jobs cancelled by TAS users

There are three main destinations for the logging information recorded by TLP; they are:

1. COINS Network Management and Monitoring System (CNMMS). This is a host in the COINS network to which magnetic tapes containing COINS Interrogation (COINI) records are sent weekly.
2. TAS Activity Log (TAL). This is a TAS local file which is a comprehensive chronological record of all significant TAS events.
3. TAS LOG (TLOG). This is the TAS operator's console upon which a subset of TAS events (error messages, network status, security violations etc.) is printed.

GLOBAL DATA USAGE - TLP uses the following global data files:

GLJOBS	-	COINI record descriptions
GQFIL	-	batch query filenames for authorization check
GLBUF	-	COINI record buffer

GTAUT	-	terminal authorization data
GTERM	-	terminal descriptions
GUDES	-	user descriptions
LTPID	-	logging tape daemon process ID
TLTAL	-	TAS activity log
TLOG	-	disk buffer for TAS LOG

LOCAL DATA USAGE - TLP uses the following local data:

TLCOINI	-	COINS interrogation records
---------	---	-----------------------------

GENERAL PROCESS FLOW - The main function of each event processor is identical; only the functions which actually process the event are different and are described below under the MAJOR FUNCTION DESCRIPTIONS.

TLP first saves the time of the event and real user ID of the process for later use in indexing GUDES. The user ID is then set to that of TAS (TUID) so that inter-process communication with the Logging Tape Daemon (LTD) can occur. TLP then builds a TAL header block and opens the TAL and TLOG for appending. The function TLP-EVENT is called to process the specific event. Upon completion of the call, TLP closes TAL and TLOG, unlocks TAL, GLJOBS and TLOG, then terminates. See figure 65 for data flow.

MAJOR FUNCTION DESCRIPTIONS - The following paragraphs relating to functions TLP-EVENT 00-13 describe the event specific processing of the TLP processes TLP00 - TLP13.

TLP-EVENT 00 (TAS Logon) - TLP00 is called when a user makes an (un)successful attempt to log onto TAS. The TAL is first locked, a record is formatted and appended to it and the TAL is unlocked. If the logon was unsuccessful, a COINI record is initialized and formatted for an "unsuccessful TAS logon attempt". The function TLCNMMS is called to write the record to magnetic tape and TLP-EVENT 00 makes a normal return.

TLP-EVENT 01 (TAS Logoff) - TLP01 is called when a user logs off of TAS. The TAL is locked, a record is formatted and appended to it and the TAL is then unlocked. TLP-EVENT 01 then makes a normal return.

TLP-EVENT 02 (User Submits Batch Query) - TLP02 is called when a user has submitted a batch query to TAS and the query has been validated, becoming a valid TAS job. First the TAL is locked and the query file names are read from the global file GQFIL in the user's directory. A record for this event is then formatted and appended to the TAL, which is then unlocked. The function TLJOB is called to enter this job into the logging job directory GLJOBS. A COINI record is then initialized and formatted for this event and written out to a disk file for later updates. TLP-EVENT 02 then makes a normal return.

TLP-EVENT 03 (RCPT Received or Not Received for a Batch Query) - TLP03 is called when TAS receives (or doesn't receive) a RCPT message from the distant batch host for a specific query. First, the TAL is locked, a record for this event is formatted and appended to it and the TAL is unlocked. If no COINI record for this query exists, TLP-EVENT 03 takes an error return. Otherwise, the COINI record file is read in. If the RCPT was received, the COINI record is modified for the DTG of the receipt and

written back out to disk. If no RCPT was received, the DTG of the query logout is recorded in the COINI record and it is written onto magnetic tape. TLP-EVENT 03 then takes a normal return.

TLP-EVENT 04 (RLSE Received for a Batch Query) - TLP04 is called when TAS receives a RLSE message from the distant batch host for a specific job. First the TAL is locked, a record for this event is formatted and appended to it and the TAL is unlocked. If no COINI record exists for this job, TLP-EVENT 04 takes an error return. Otherwise, the COINI record is updated for the DTG of the RLSE receipt and written back out to disk. TLP-EVENT 04 then takes a normal return.

TLP-EVENT 05 (Response Received for a Batch Query) - TLP05 is called when TAS receives a response to a batch query job. First the TAL is locked, a record for this event is formatted and appended to it and the TAL is unlocked. If no COINI record exists for this job, TLP-EVENT 05 takes an error return. Otherwise, the COINI record is read in. If the response was an ANSR (answer), the COINI record is updated and written back out to disk. If the response was not an ANSR, a test is made if the response was "No reply". If so, the COINI record is updated for "no reply received for batch query" and written out to magnetic tape. If not, the COINI record is updated for "ABORT response received" and the record is written back out to disk. TLP-EVENT 05 then takes a normal return.

TLP-EVENT 06 (Non-existent JOBID from the Network) - TLP06 is called when TAS receives a non-existent JOBID from the network. The TAS LOG is locked, a record is formatted for this event and appended to it and it is unlocked. TLP-EVENT 06 then takes a normal return.

TLP-EVENT 07 (Batch Query Response Display Attempt) - TLP07 is called when a user makes an (un)successful attempt to display the response to a batch query. First the TAL is locked, a record for this event is formatted and appended to it and the TAL is unlocked. If the COINI record for this job exists, it is read in; otherwise a flag is set indicating that this job has been successfully displayed at least once, and a new COINI record is initialized. If the user was not authorized to view the response, the TAS LOG is locked, a record for this event is formatted and written to it and TLOG is unlocked. The COINI record is then updated for "batch query not displayed". If the user was authorized to view the response, a test is made to determine if the terminal was authorized for the response. If so, the COINI record is updated for "batch query response displayed". Otherwise, the TAS LOG is locked, a record is formatted for this event and appended to it and it is unlocked. Finally, if this was a display of a response viewed successfully at least once, or the user was not authorized to view the response, the COINI record is written out to magnetic tape. TLP-EVENT 07 then takes a normal return.

TLP-EVENT 08 (Batch Query Job Cancelled) - TLP08 is called when either the user or a TAS internal process causes a batch query job to be cancelled. First, if the COINI record for this job does not exist, TLP-EVENT 08 takes an error return. Otherwise, the COINI record file is read in. The TAL is then locked, a record for this event is appended to it and the TAL is unlocked. The TAS LOG is then locked, a record is formatted and output to it, and it is unlocked. Finally, the COINI record is updated for the DTG of the query cancellation and written out to magnetic tape, and TLP-EVENT 08 takes a normal return.

TLP-EVENT 09 (SRVC TRACE Returns Non-Existent JOBID) - TLP09 is called when a SRVC TRACE of a specified JOBID returns a status indicating that the distant host has no record of that job. First, the TAL is locked, a record is formatted and appended to it, and it is unlocked. If the COINI record for this job does not exist, TLP-EVENT 09 takes an error return; otherwise, the COINI record for this job is read in, updated for the DTG of query logout and written onto magnetic tape. TLP-EVENT 09 then takes a normal return.

TLP-EVENT 10 ((Un)successful Interactive Host Logon Attempt) - TLP10 is called when a user requests a session with a remote interactive host and TAS performs the transparent distant host logon sequence. First the TAL is locked, a record for this event is formatted and appended to it and it is unlocked. A COINI record is then created, and if the logon was successful, the record is updated for a successful logon, an entry is made in the logging directory file GLJOBS for the COINI record, and the record is written out to disk. If the logon attempt was unsuccessful, the COINI record is so updated and written to magnetic tape. Also, the TAS LOG is locked, a record is formatted and appended to it, and the TLOG is unlocked. Finally, TLP-EVENT 10 takes a normal return.

TLP-EVENT 11 (Interactive Host Logoff) - TLP11 is called when the user logs off of an interactive host thus terminating the session. First the TAL is locked, a record for this event is formatted and appended to it and it is unlocked. If the COINI record for this job does not exist, TLP-EVENT 11 takes an error return. Otherwise, the COINI record file is read in, updated for the completion of an interactive session and written out to magnetic tape. TLP-EVENT 11 then takes a normal return.

TLP-EVENT 12 (Unauthorized Application Access Attempt) - TLP12 is called when a TAS user attempts to access an application which is not authorized to either him or his terminal. First, the TAL is locked, a record is formatted and appended to it and it is unlocked. The TAS LOG is then locked, a record is formatted and appended to it and the TLOG is unlocked. Finally, a COINI record for "unauthorized application access attempt" is created and written to magnetic tape. TLP-EVENT 12 then takes a normal return.

TLP-EVENT 13 (Unauthorized Batch File Access Attempt) - TLP13 is called when a TAS user attempts to query a batch host file to which either he or his terminal is not entitled access. First, the TAL is locked, the query file names and access statuses are read from the global file GQFIL in the user's directory, a record is formatted and appended to the TAL, and it is unlocked. Then, the TAS LOG is locked, a record is formatted and output to it for this event and the TLOG is unlocked. Finally, a COINI record for "unauthorized file access attempt" is created and written out to magnetic tape. TLP-EVENT 13 then takes a normal return.

TLCNMMS (Write a COINI Record to Magnetic Tape) - TLCNMMS is the interface routine between TLP and the TAS logging tape on which the COINI records are collected for transmission to CNMMS. First, if the COINI record lacks a JOBID (in the case of security violations), GFJOBID is called to supply a logging ID to the record. If the disk file copy of the COINI record is to be deleted, the logging directory GLJOBS is opened and searched for a match on the input JOBID. If a match is found, the specified disk file is unlinked, the GLJOBS item is marked as "free" and GLJOBS is closed. GLJOBS is then unlocked and GLBUF is locked, opened and the number of

COINI records currently in the buffer is read. If the file was empty, the number of records is set to zero. The number of records is then incremented and written back out to GLBUF. The input JOBID is written into the appropriate slot in the GLBUF header and the COINI record is written into the appropriate block of GLBUF. If there are less than ten records in the buffer, GLBUF is unlocked. Otherwise, the buffer contains a physical tape record. TLCNMMS retrieves the PID of the Logging Tape Daemon from LTPID and a signal is sent to LTD to copy the records out to tape. TLCNMMS takes a normal return.

TLICR (Initialize a COINI Record) - When called from TLP event processors 00, 02, 07, 10, 12 and 13, TLICR first blanks out the buffer to be used for the in-core COINI record. The host ID, addressee and terminal ID fields are then set to their default values. The query filenames are set to "UNK" (for unknown) and the security flag fields are all set to 'Y'. TLICR then locks GLJOBS (for later access to the COINI record by TLWCR and TLCNMMS) and takes a normal return.

TLJOB (Make a GLJOBS Entry for a COINI Record) - When called by the TLP event processors 02 and 10, TLJOB first locks and opens GLJOBS. GLJOBS items are read until either an unoccupied item is found or the end-of-file is reached. A GLJOBS item for the new COINI record is then written out, GLJOBS is closed and unlocked and TLJOB takes a normal return.

TLRCR (Read a COINI Record File) - When called by TLP event processors 03, 04, 05, 07, 08, 09 and 11, TLRCR first builds the COINI record file pathname and opens the file. If the file does not exist, TLRCR returns an error code of -1. Otherwise, GLJOBS is locked for access to the record, the COINI record file is read and closed, and TLRCR returns a value of 0.

TLWCR (Write a COINI Record File) - When called by TLP event processors 02, 03, 04, 05 and 10, TLWCR first builds the COINI record file pathname and creates the file. The COINI record file is then written out and closed, GLJOBS is unlocked and TLWCR takes a normal return.

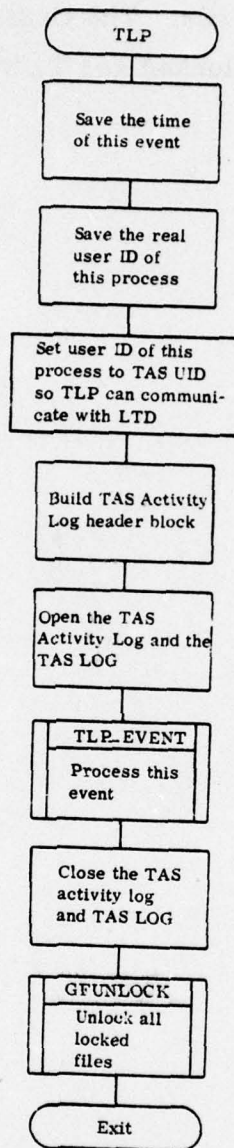


Figure 65. TLP Process Data Flow (Sheet 1 of 20)

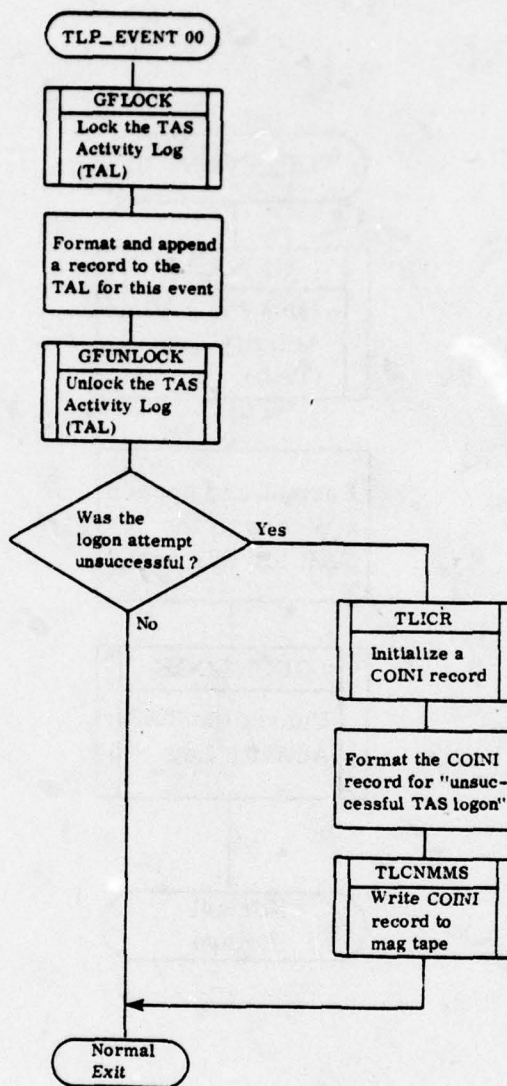


Figure 65. TLP Process Data Flow (Sheet 2 of 20)

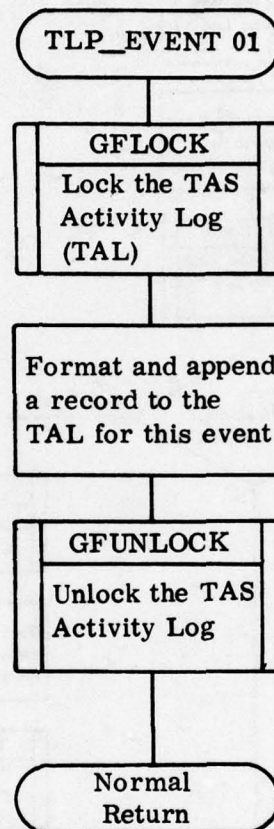


Figure 65. TLP Process Data Flow (Sheet 3 of 20)

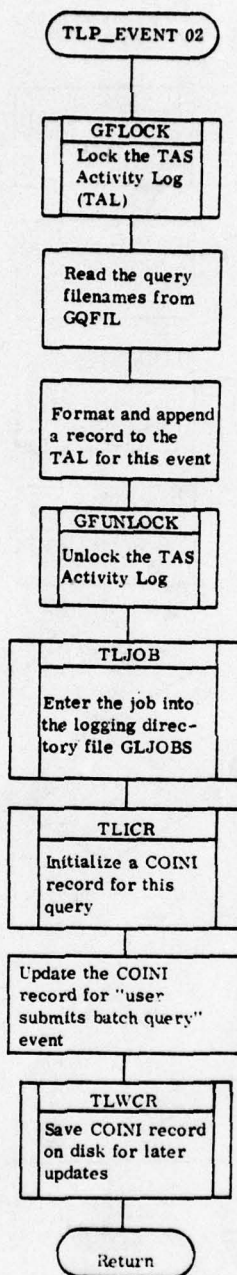


Figure 65. TLP Process Data Flow (Sheet 4 of 20)

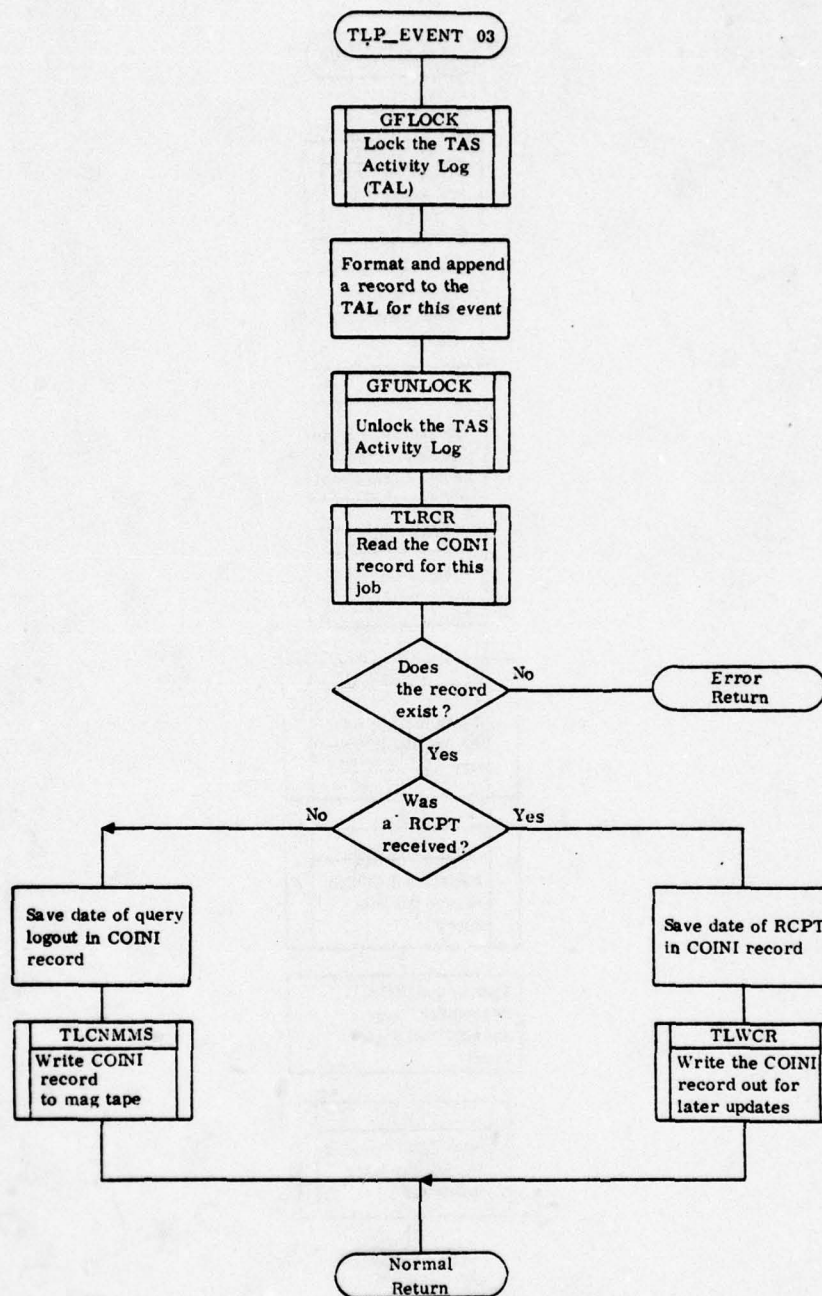


Figure 65. TLP Process Data Flow (Sheet 5 of 20)

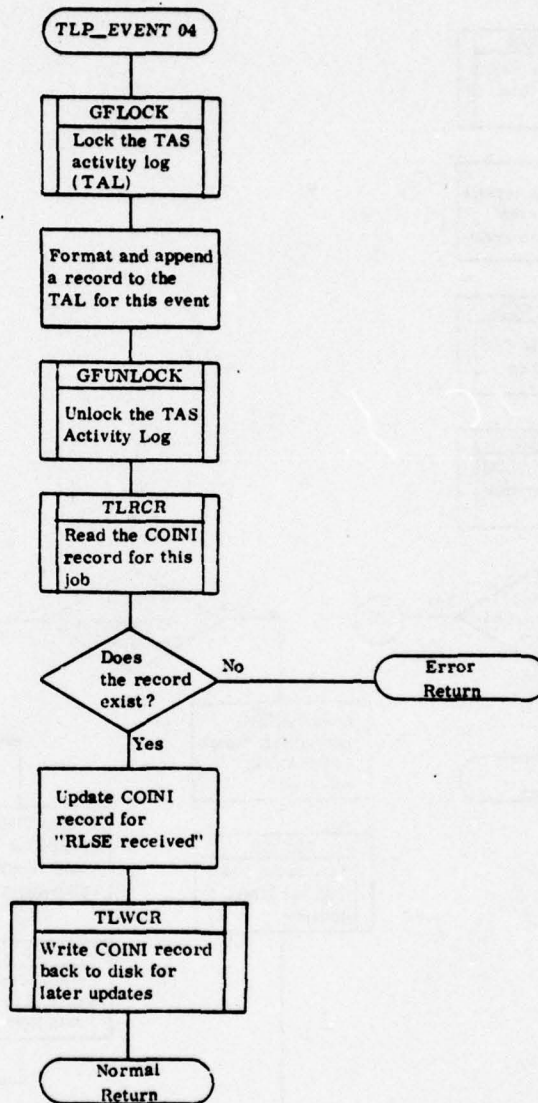


Figure 65. TLP Process Data Flow (Sheet 6 of 20)

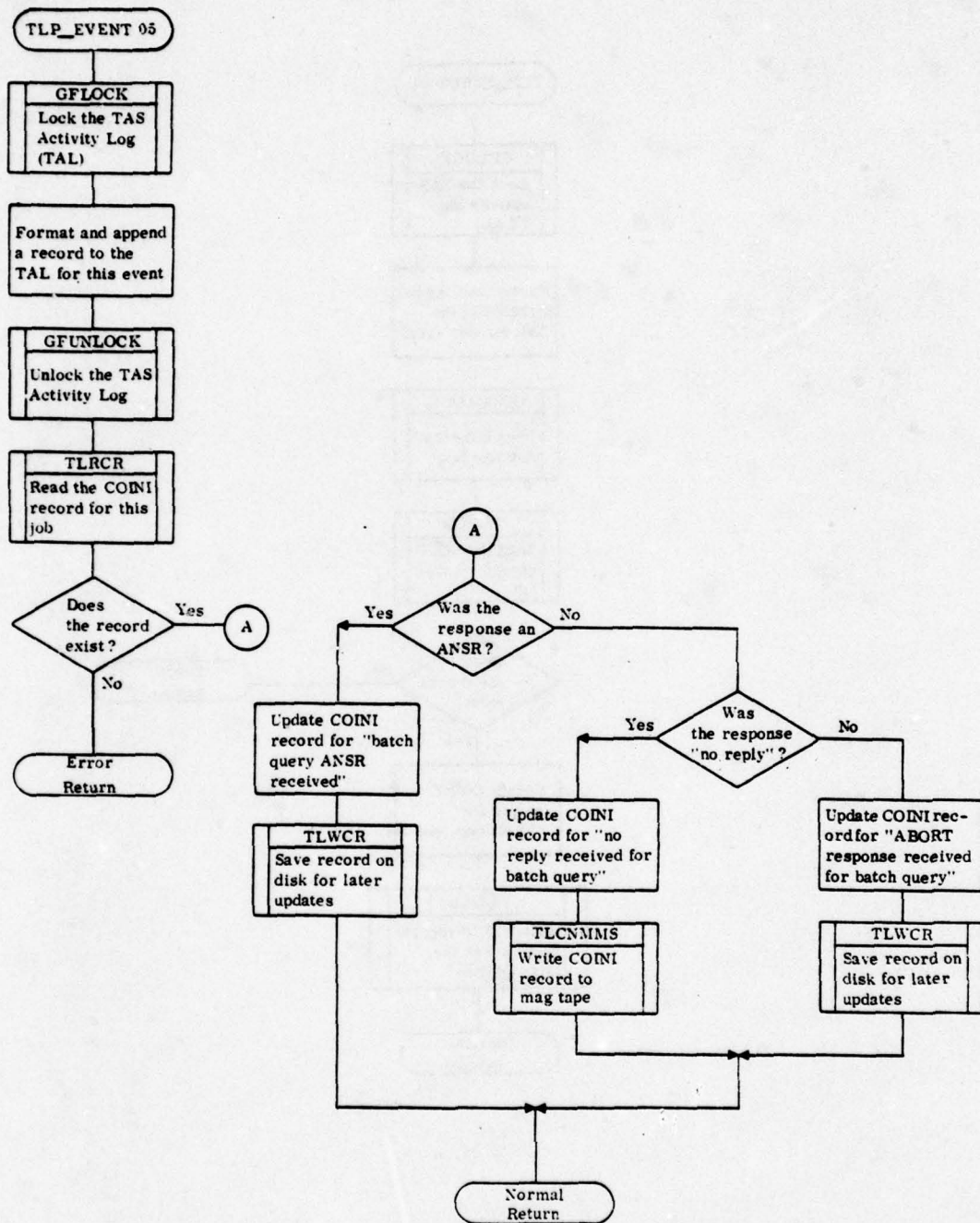
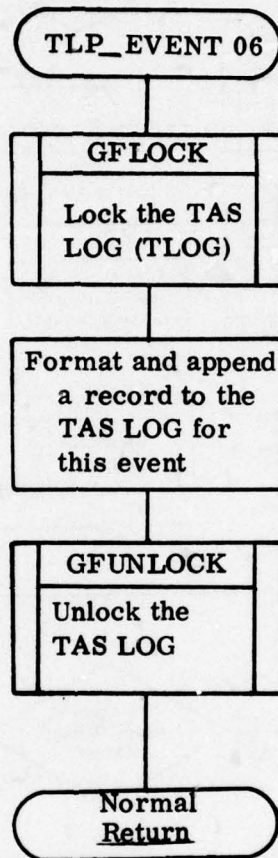


Figure 65. TLP Process Data Flow (Sheet 7 of 20)



• Figure 65. TLP Process Data Flow (Sheet 8 of 20)

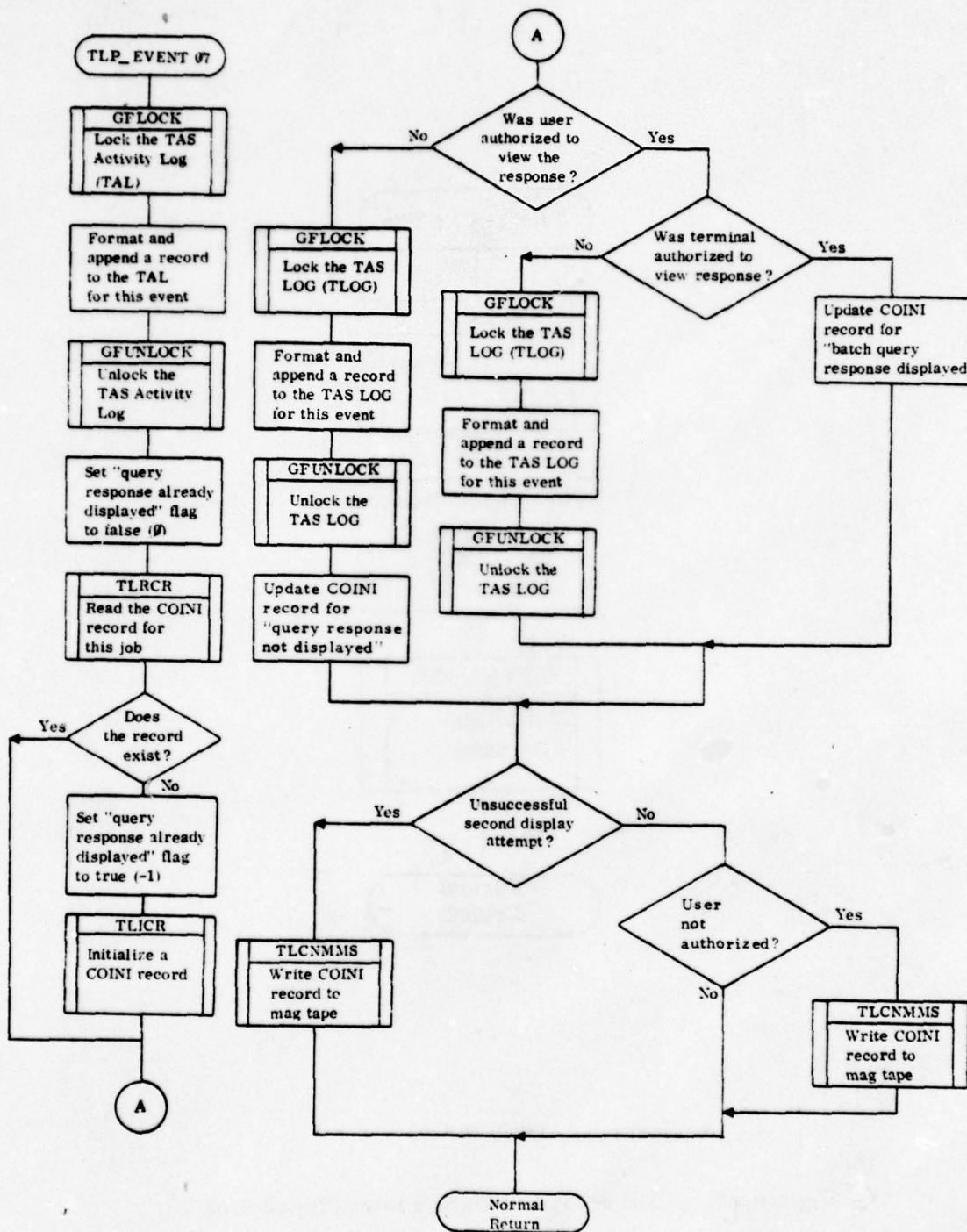


Figure 65. TLP Process Data Flow (Sheet 9 of 20)

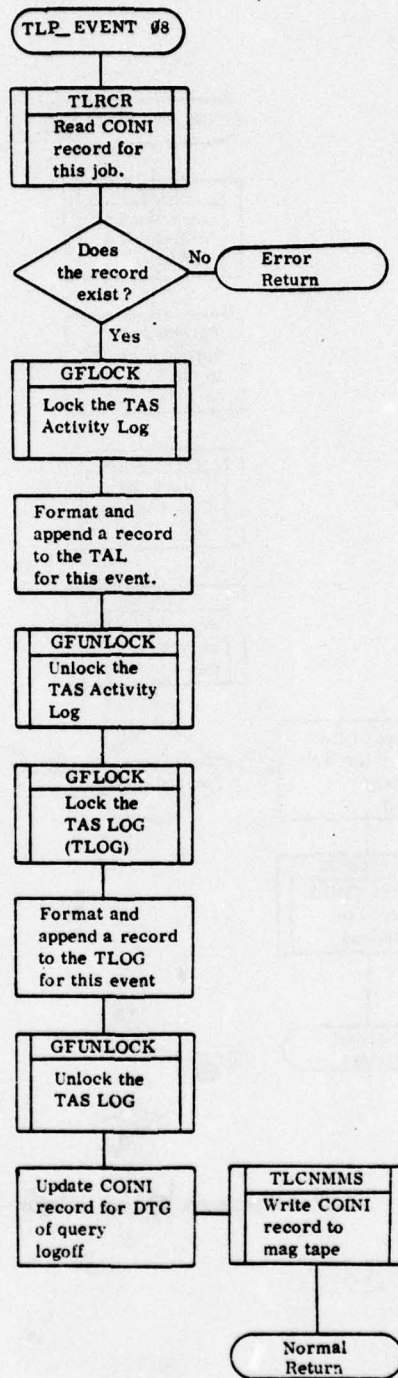


Figure 65. TLP Process Data Flow (Sheet 10 of 20)

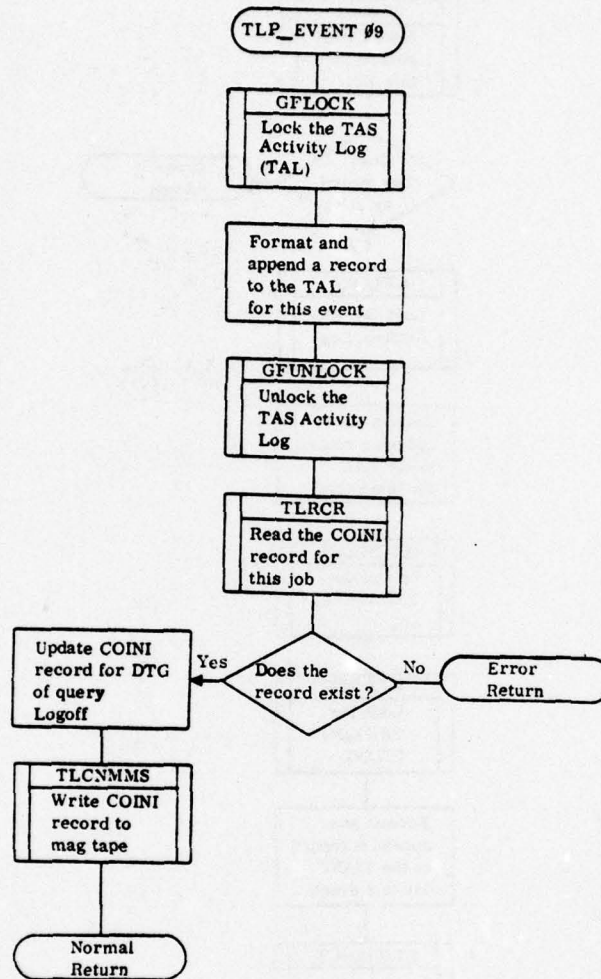


Figure 65. TLP Process Data Flow (Sheet 11 of 20)

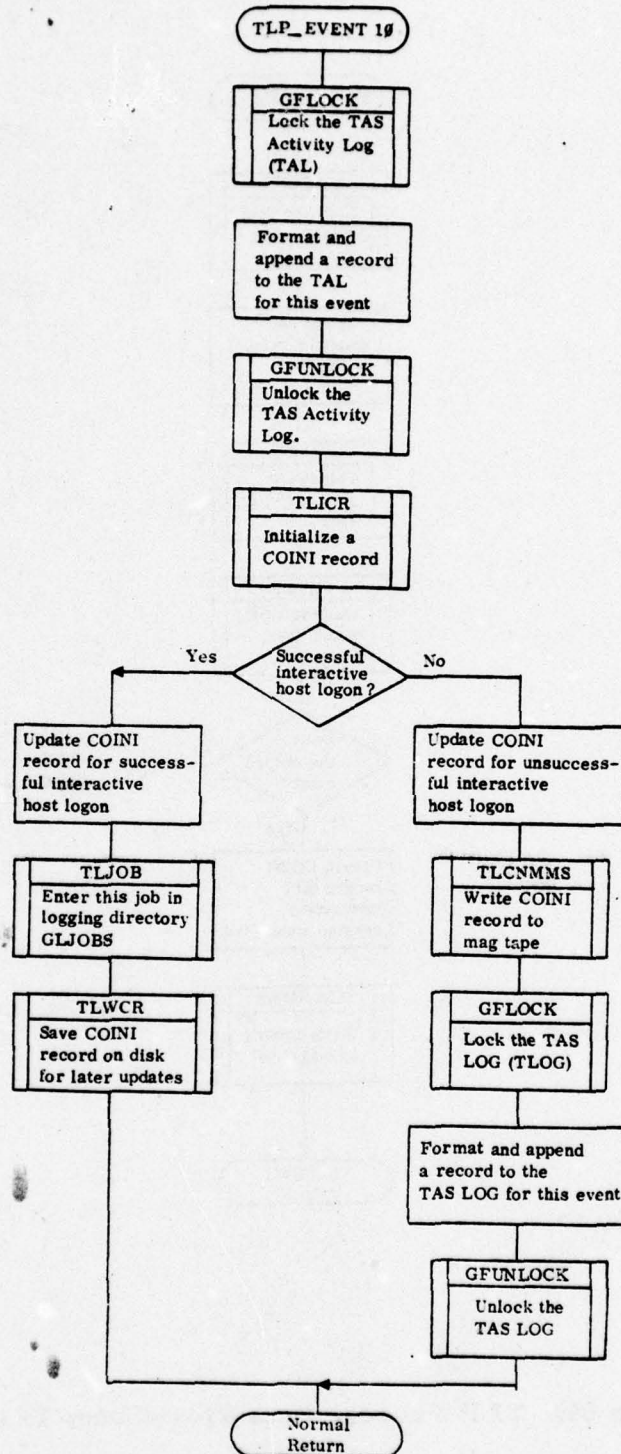


Figure 65. TLP Process Data Flow (Sheet 12 of 20)

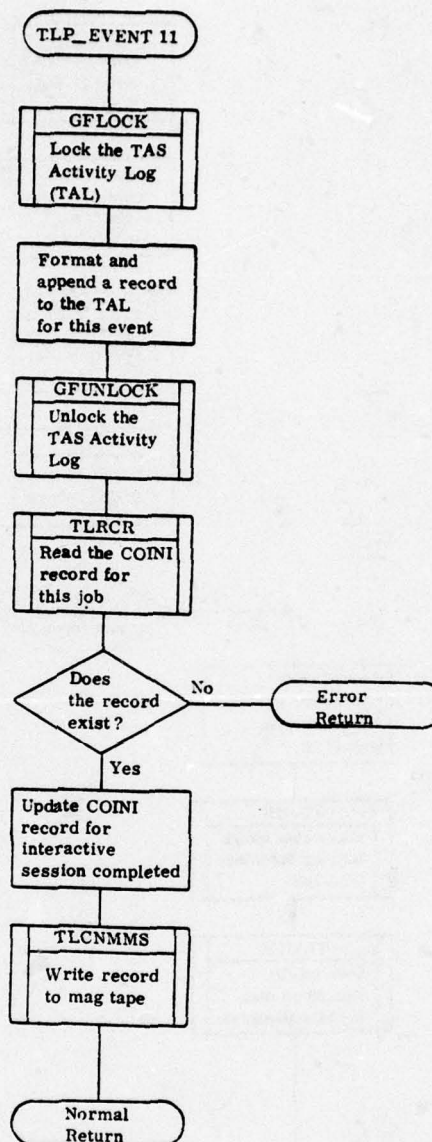


Figure 65. TLP Process Data Flow (Sheet 13 of 20)

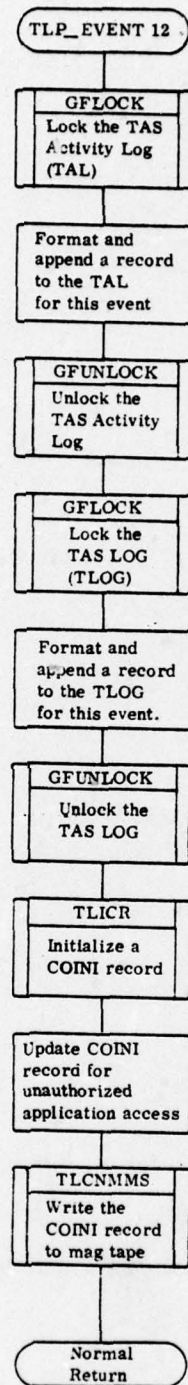


Figure 65. TLP Process Data Flow (Sheet 14 of 20)

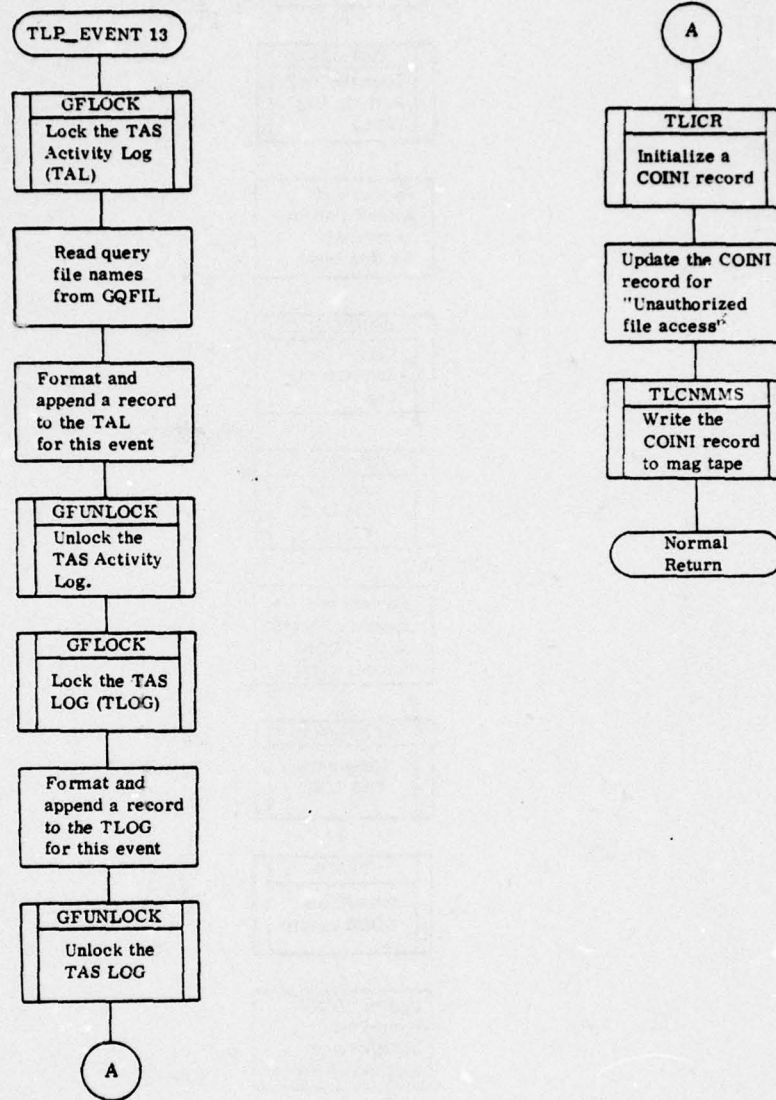


Figure 65. TLP Process Data Flow (Sheet 15 of 20)

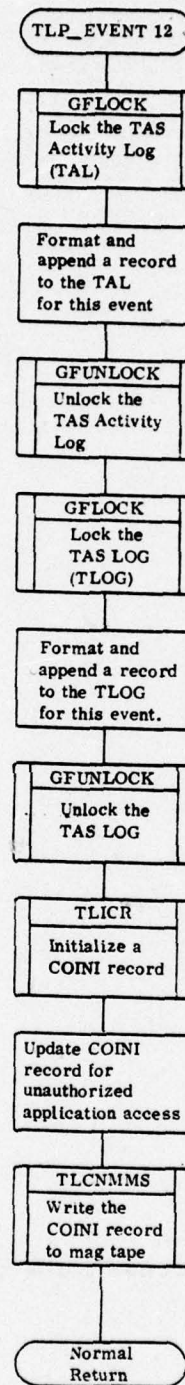


Figure 65. TLP Process Data Flow (Sheet 14 of 20)

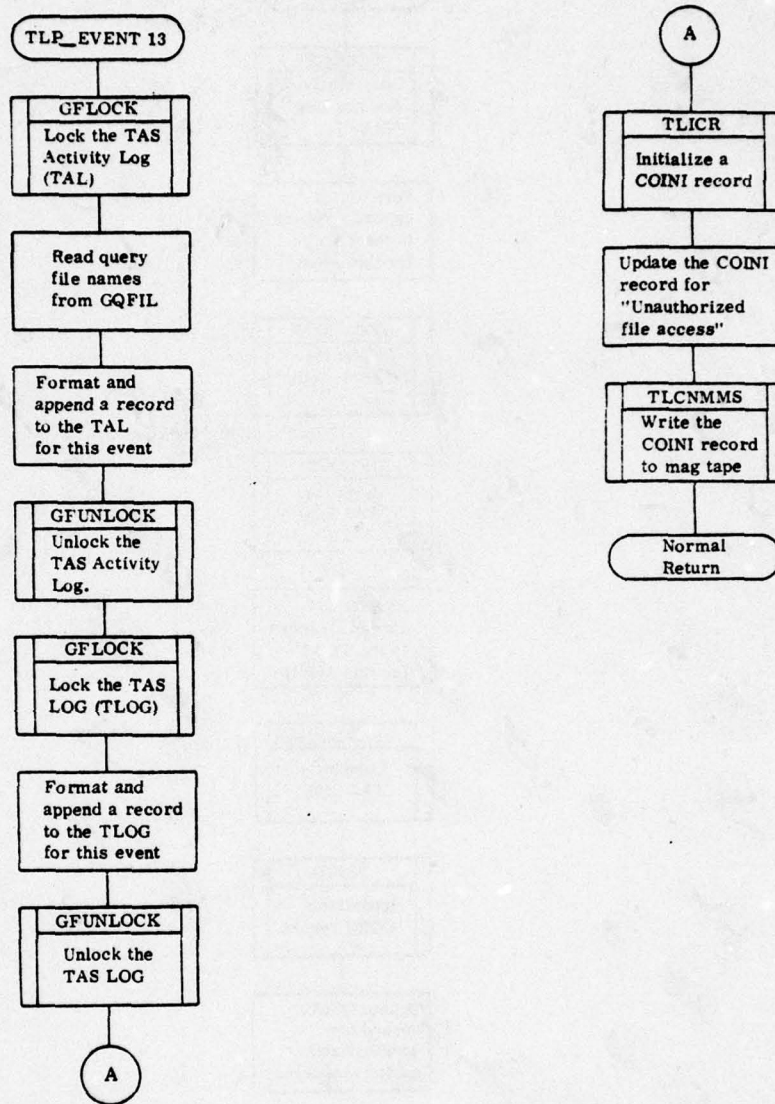


Figure 65. TLP Process Data Flow (Sheet 15 of 20)

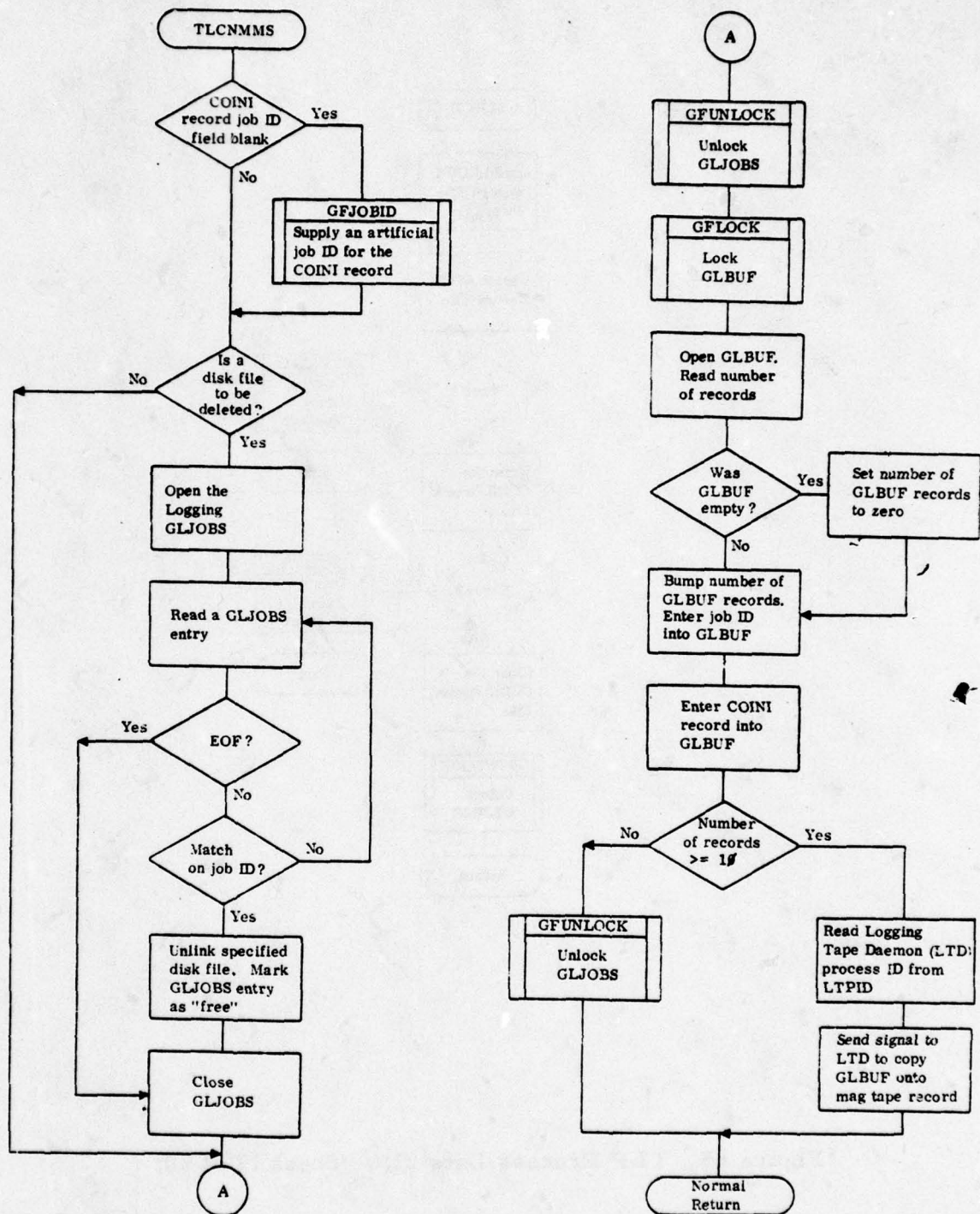


Figure 65. TLP Process Data Flow (Sheet 16 of 20)

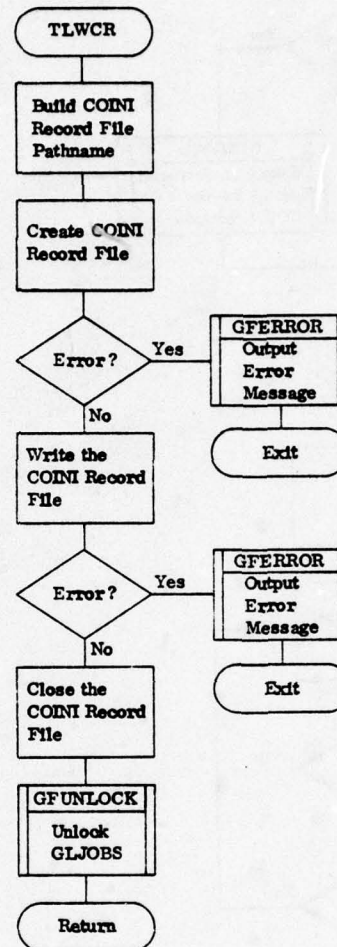


Figure 65. TLP Process Data Flow (Sheet 17 of 20)

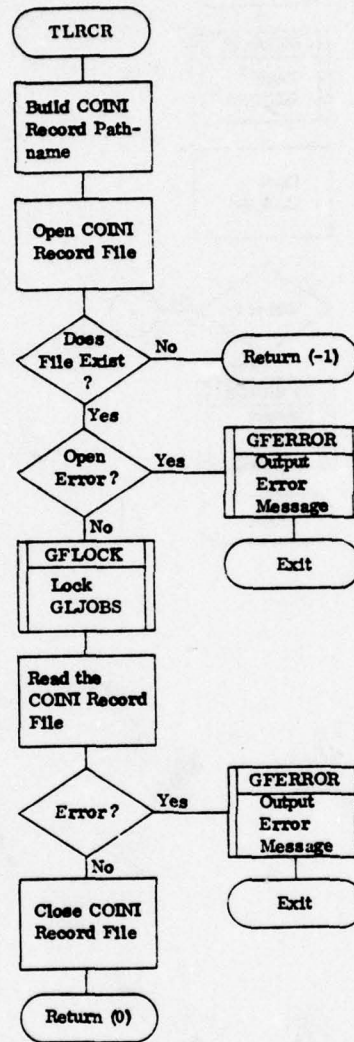


Figure 65. TLP Process Data Flow (Sheet 18 of 20)

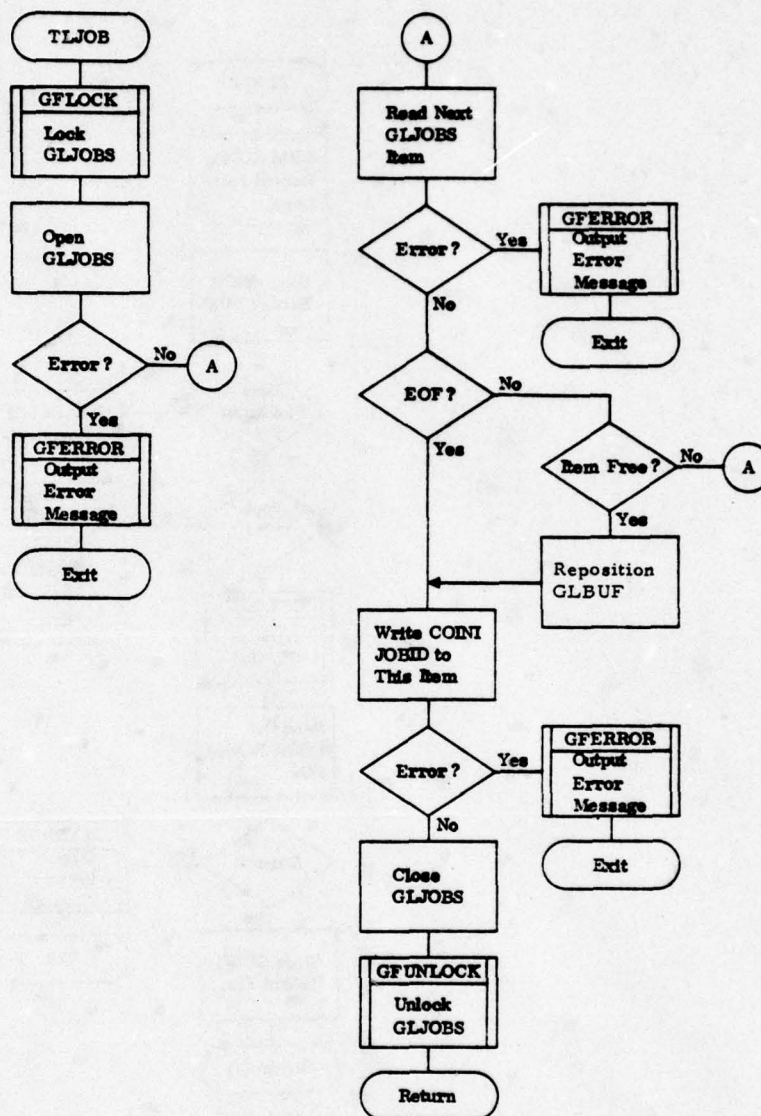


Figure 65. TLP Process Data Flow (Sheet 19 of 20)

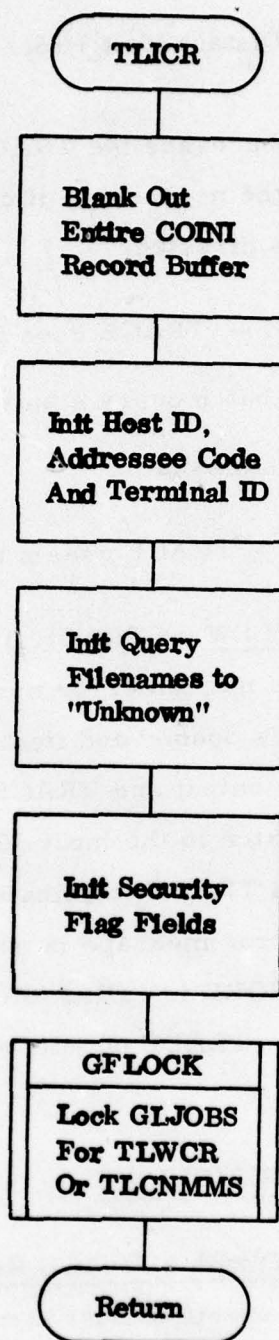


Figure 65. TLP Process Data Flow (Sheet 20 of 20)

TRACE (Trace the Distant Host Processing of a Batch Query)

The TRACE process processes the TRACE command. It allows a user to determine the status of the processing of one of his batch queries by that host to which the query was directed.

GLOBAL DATA USAGE - TRACE uses the following global data:

GBQST	-	user batch query statuses
GHOST	-	host statuses

LOCAL DATA USAGE - TRACE uses a temporary user text file.

GENERAL PROCESS FLOW - TRACE first checks to see if a JOBID argument has been input. If not, an error message is output and TRACE terminates. Next, GBQST is opened and item zero is read. If the user has no batch jobs, a message is output and TRACE terminates. Otherwise, GBQST is searched for a match to the input JOBID. If no match is found, an error message is output and TRACE terminates. If a match is found, but the job is not pending, an error message is output and TRACE terminates. Otherwise, the function TRBQRD is called to call the BQRD process with the specified trace function. TRACE outputs a message and terminates. See figure 66 for data flow.

MAJOR FUNCTION DESCRIPTIONS

TRBQRD (Execute BQRD with a TRACE Request) - TRBQRD first opens and reads GHOST and if the specified host is not up, an error message is output and TRBQRD exits. If the host is up, GFJOBID is called to obtain a new TAS JOBID, a temporary JOBID file is created and the JOBID to be traced is

written to it. The BQRD process is then called via a fork/execute sequence. If the TRACE message was not sent, a message is output and TRBQRD exits. Otherwise, TRBQRD returns.

TRERROR (General Error/Exit Function) - TRERROR first checks if a message number was passed as an argument. If so, GFERROR is called to output the specified message. If the process is not to exit, TRERROR returns. Otherwise, the temporary output file is unlinked, GFERROR is called to output an end message and TRERROR exits.

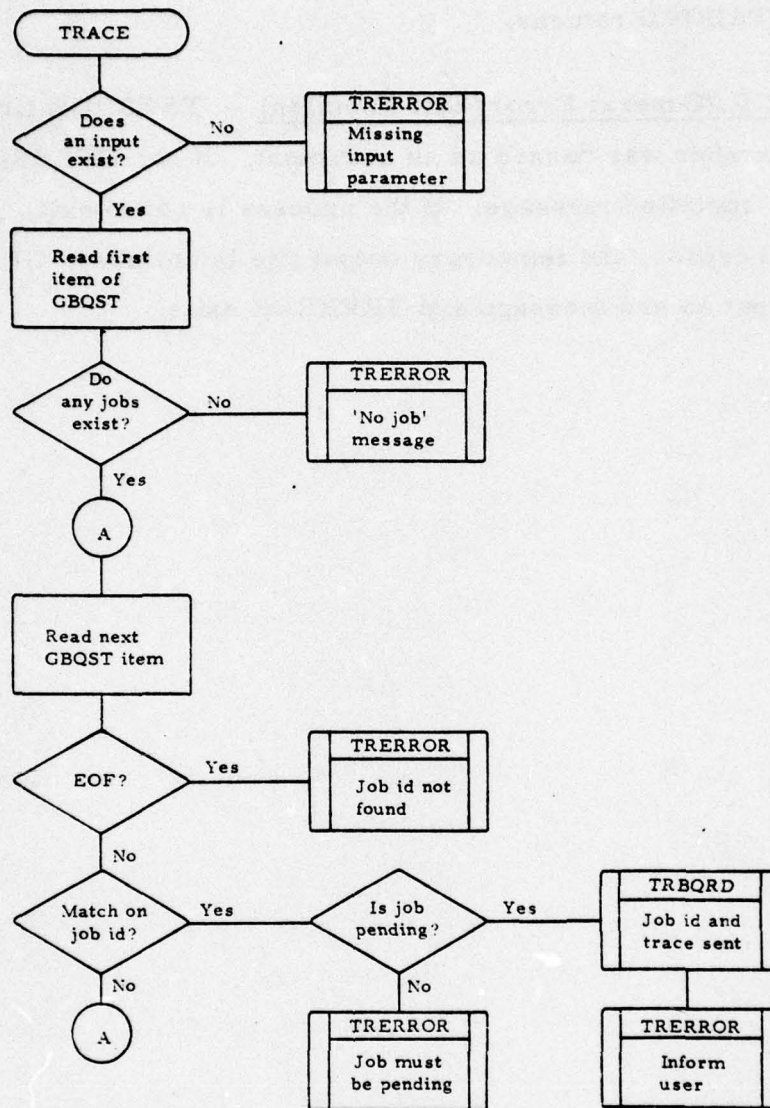


Figure 66. TRACE Process Data Flow (Sheet 1 of 2)

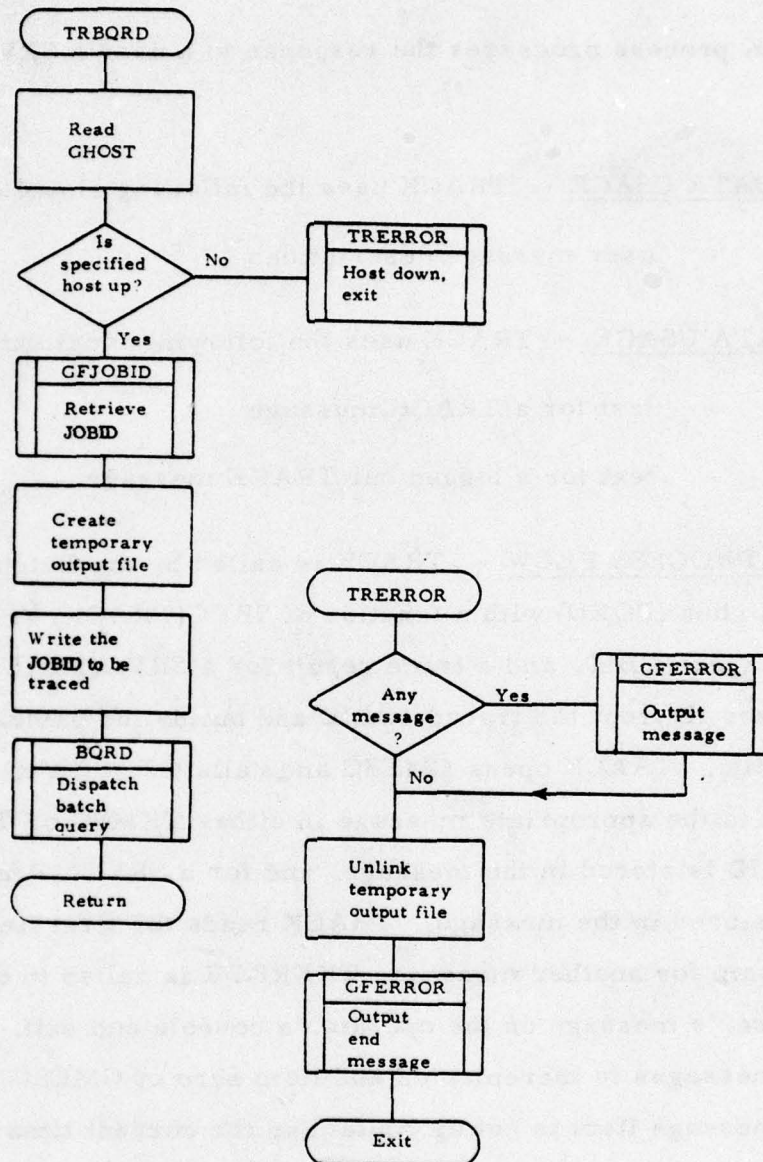


Figure 66. TRACE Process Data Flow (Sheet 2 of 2)

TRACK (Process a User's TRACE Response)

The TRACK process processes the response to a user's SRVC(TRACE) request.

GLOBAL DATA USAGE - TRACK uses the following global data:

GMESG - user message descriptions

LOCAL DATA USAGE - TRACK uses the following local data:

TKMSG - text for a TRACK message

TKLMG - text for a logged out TRACE message

GENERAL PROCESS FLOW - TRACK is called by the Batch Query and Response Dispatcher (BQRD) with a function of SRVC(TRACK) or SRVC(TRACE) logged out, a traced JOBID, and a trace result for a SRVC(TRACK). TRACK retrieves the user ID from the traced JOBID and builds the pathname for the user's GMESG file. TRACK opens GMESG and calls GFLOCK to lock GMESG. A pointer is set to the appropriate message in either TKMSG or TKLMG. The traced JOBID is stored in the message, and for a SRVC(TRACK), the trace result is stored in the message. TRACK reads the first item of GMESG. If there is no room for another message, TKERROR is called to output an error and the user's message on the operator's console and exit. Otherwise, the number of messages is incremented and item zero of GMESG is written out. The new message item is set up containing the current time and an invalid originator code (-1). The new GMESG item is written out. A message file is created, the appropriate message text is written out to the file, and the file is closed. TRACK calls TKERROR to exit. See figure 67 for data flow.

MAJOR FUNCTION DESCRIPTIONS

TKERROR (Output Error and/or Exit) - TKERROR is called with an error number or zero, and an exit flag or zero. If there is an error to output, GFERROR is called to output the error. If the exit flag is not set, TKERROR returns. Otherwise, if GMESG is locked, GFUNLOCK is called to unlock it, and TKERROR exits.

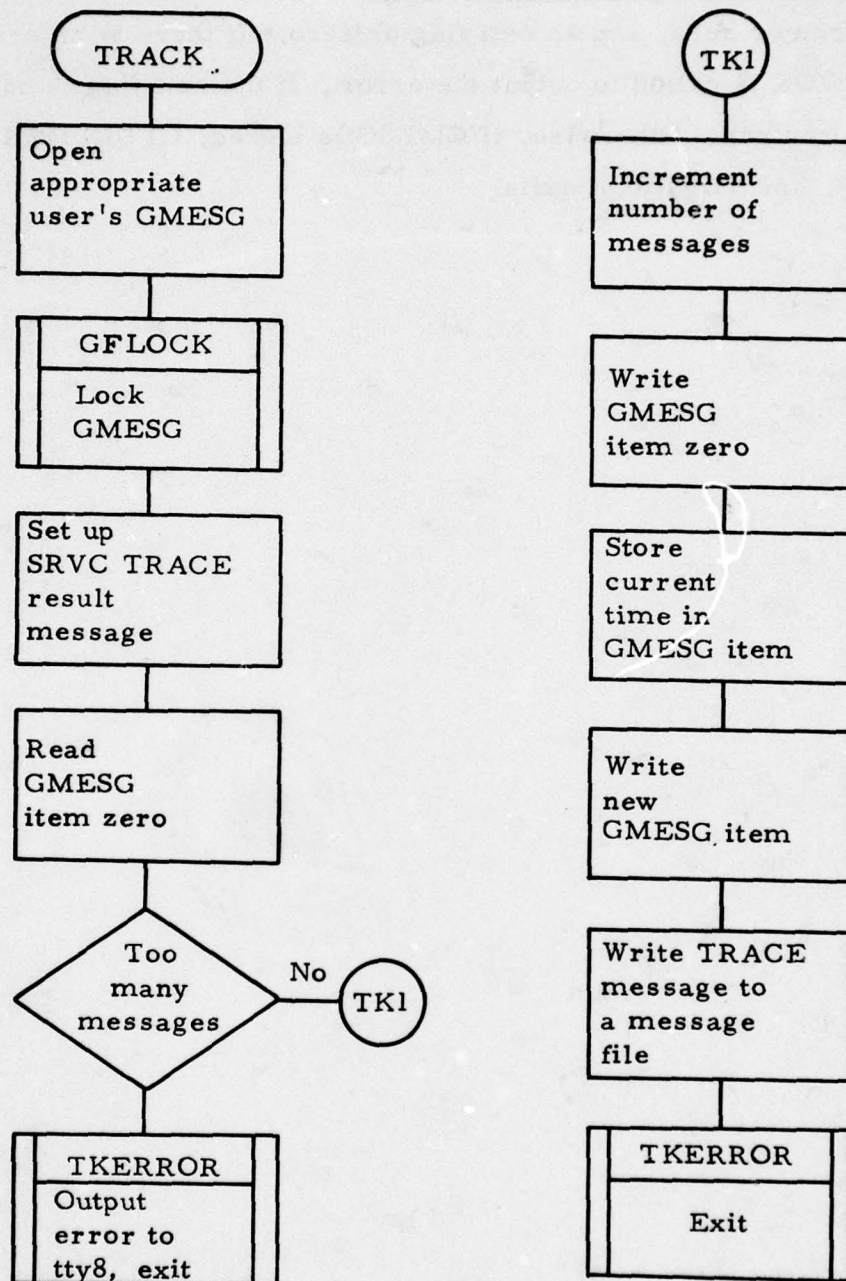


Figure 67. TRACK Process Data Flow (Sheet 1 of 2)

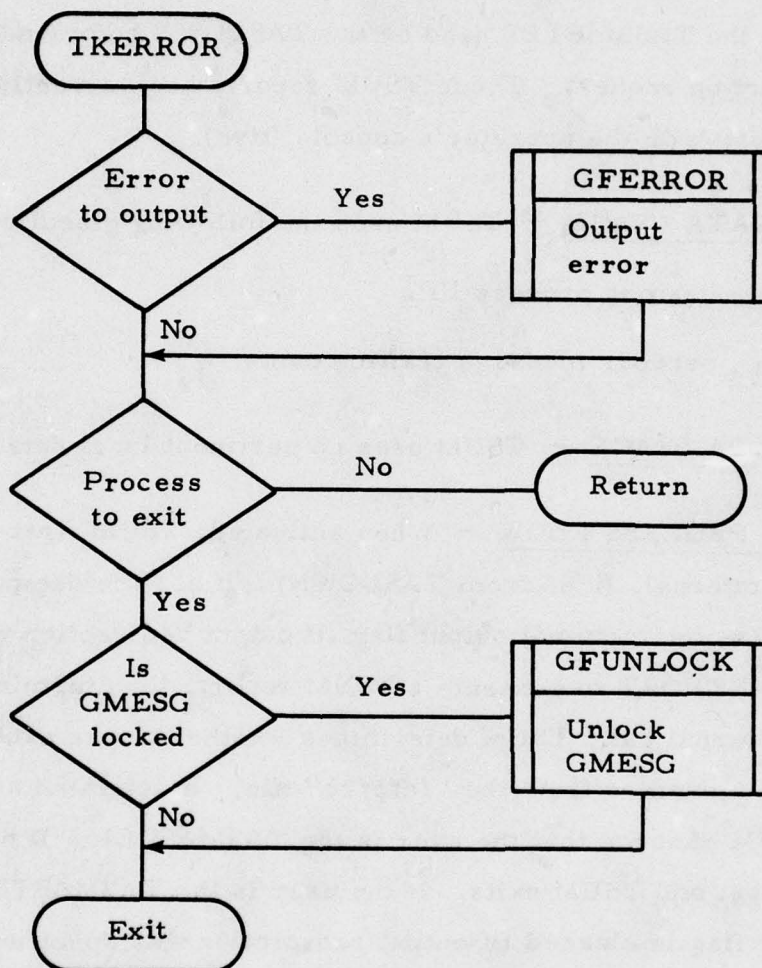


Figure 67. TRACK Process Data Flow (Sheet 2 of 2)

TSUM (Output a Traffic Summary Report)

TSUM is activated from the "/etc/rc" file during TAS initialization to produce a TAS network traffic report every six hours, or by a keyboard command from the TASMMASTER (and by the TASDOWN command) to produce the TSUM report on request. These TSUM reports are currently output in a formatted version on the operator's console (tty8).

GLOBAL DATA USAGE - TSUM uses the following global data:

GPIDS - daemon process ID's

GTCTS - network message traffic counts

LOCAL DATA USAGE - TSUM uses no pertinent local data.

GENERAL PROCESS FLOW - When activated, TSUM first determines if the call was internal, (i. e. from TASDOWN). If so, the daemon flag is set, TSUM clears the terminal output flag (if output redirection was specified), calls TS-REPORT to generate a TSUM report, then terminates. If it was not an internal call, TSUM determines whether it was executed as a command or as a daemon from the "/etc/rc" file. If activated as a keyboard command, TSUM ensures that the user is the TASMMASTER. If not, an error message is output and TSUM exits. If the user is the TASMMASTER, the terminal output flag is cleared (if output redirection was specified), TS-REPORT is called to generate the TSUM report and TSUM terminates.

If TSUM was activated as a daemon from the "/etc/rc" file, it ignores all interrupt signals and saves its process ID. GPIDS is then locked via GFLOCK, opened, and the process ID is appended to the end of the file.

GPIDS is closed and unlocked via GFUNLOCK. TSUM then sets a daemon flag and opens the operator's console as the output file. Next, GTCTS is locked via GFLOCK, recreated, closed, and unlocked via GFUNLOCK. Finally, TSUM calls TS-REPORT every six hours to generate a TSUM report. See figure 68 for data flow.

MAJOR FUNCTION DESCRIPTIONS

TS-REPORT (Generate a TSUM Report) - When called, TS_REPORT saves the time of the call, then determines if TSUM was executed as a command or a daemon. If executed as a daemon, the GTCTS buffer is zeroed out and GTCTS is locked via GFLOCK. GTCTS is then opened and read. If TSUM is a daemon, GTCTS is reinitialized, zeroing out the counters, and it is then unlocked via GFUNLOCK. The TSUM record buffer itself is blanked out and the network message counts in GTCTS are converted to ASCII decimal format in the TSUM record buffer. Finally, the TSUM record is formatted and output as a report on the operator's console.

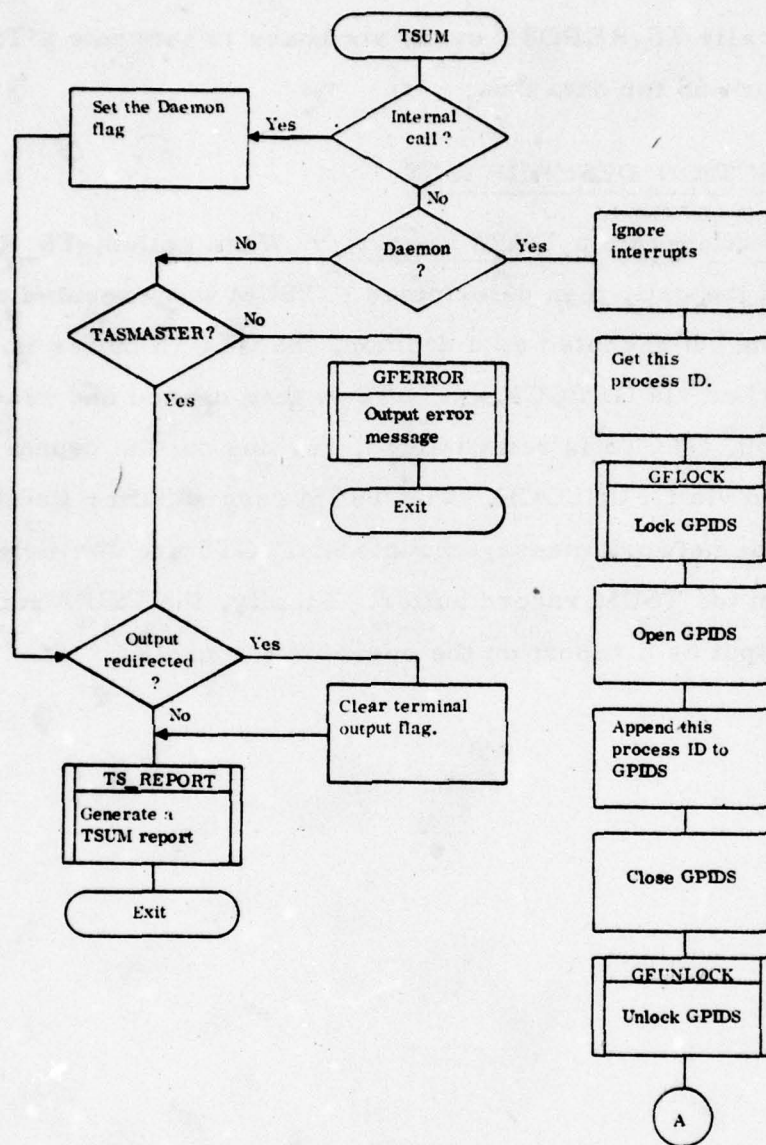


Figure 68. TSUM Process Data Flow (Sheet 1 of 2)

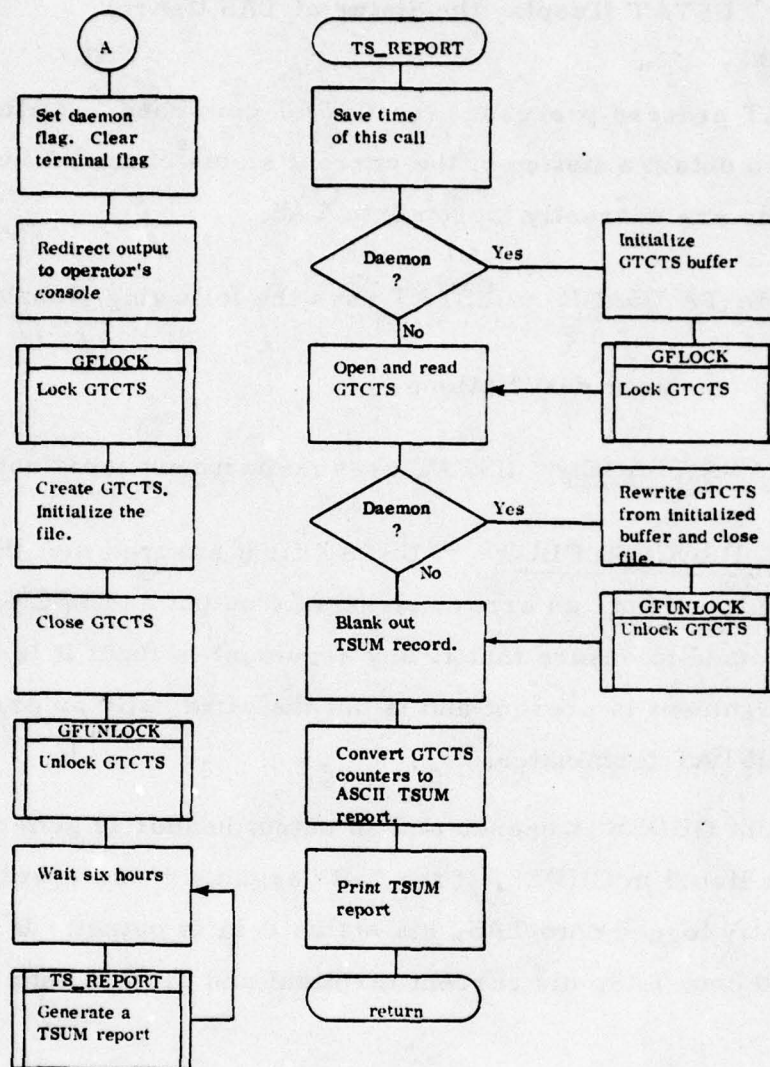


Figure 68. TSUM Process Data Flow (Sheet 2 of 2)

USTAT (Display the Status of TAS Users)

The USTAT process processes the USTAT command. It allows the TASMMASTER to obtain a listing of the current status of all TAS users or just those users who are currently logged onto TAS.

GLOBAL DATA USAGE - USTAT uses the following global data:

GUDES - user descriptions

LOCAL DATA USAGE - USTAT uses no pertinent local data.

GENERAL PROCESS FLOW - USTAT first ensures that the user is the TASMMASTER. If not, an error message is output and USTAT terminates. Next a test is made to ensure that if any argument is input it is the word "all". If an argument is present and is not the word "all" an error message is output and USTAT terminates.

At this point GUDES is opened and an output header is generated. For each TAS user listed in GUDES, if the "all" argument was specified or the user is currently logged onto TAS, his status data is output. If the user is actually logged onto TAS, his current terminal and project data are also output.

When all TAS users have been processed, USTAT terminates. See figure 69 for data flow.

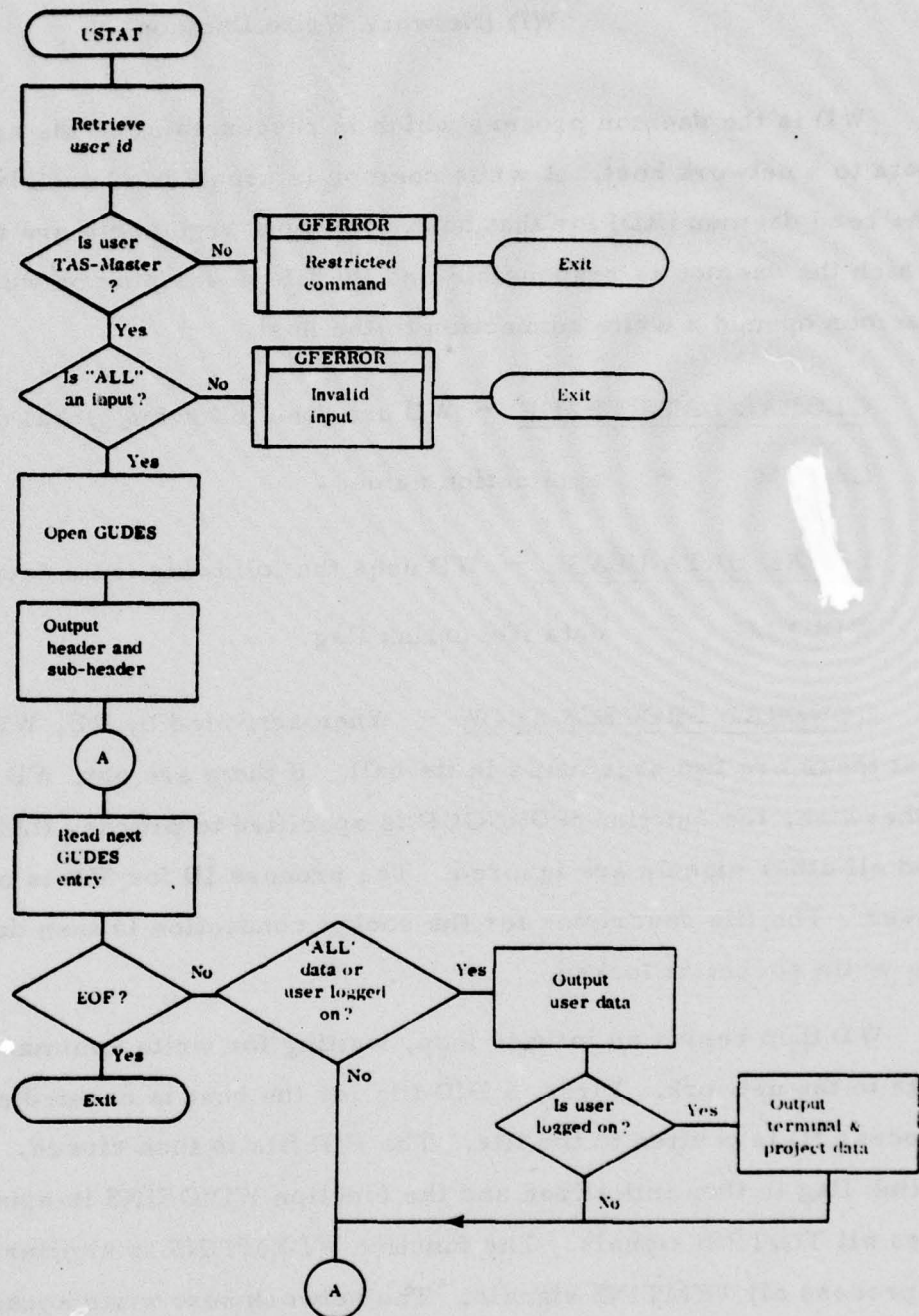


Figure 69. USTAT Process Data Flow

WD (Network Write Daemon)

WD is the daemon process which is responsible for the actual output of data to a network host. A write daemon is created for each batch host by the read daemon (RD) for that host. Its input arguments are the host ID for which the daemon is responsible and the file descriptor on which the read daemon opened a write connection to the host.

GLOBAL DATA USAGE - WD uses the following global data:

GAPNM - application names

LOCAL DATA USAGE - WD uses the following local data:

WDUFLG - data file unlink flag

GENERAL PROCESS FLOW - When activated by RD, WD first ensures that there are two arguments in its call. If there are not, WD simply exits. Otherwise, the function WDHNGUP is specified to process HANGUP signals and all other signals are ignored. The process ID for WD is obtained and saved. The file descriptor for the socket connection is then duplicated and the write socket is locked.

WD then begins an infinite loop, waiting for write commands and writing data to the network. First, a PID file for the host is created and the WD process ID is written to the file. The PID file is then closed. The data file unlink flag is then initialized and the function WDIOTINS is specified to process all TIOTINS signals. The function WDEMTINS is similarly specified to process all TEMENTINS signals. The network host write socket is then unlocked.

WD then sleeps. If a write signal is received, the JOB file (created by INI) is opened and the data file pathname is read from it. The JOB file is then closed and the data file is opened. If there was an open error, WD begins its infinite loop again.

Data is then read from the data file and written to the network via the host connection until the end-of-file is read or an error occurs. If the end-of-file is read without error, the data file is closed and unlinked (if the unlink flag was set) and WD begins its infinite loop again. If a write error to the host socket occurs, then the data file is unlinked (if the unlink flag was set), the PID file for the host is unlinked (informing INI that the connection is closed), the write socket for this host is unlocked and WD exits. See figure 70 for data flow.

MAJOR FUNCTION DESCRIPTIONS

WDHNGUP (Process Hangup Signals) - When WD receives a hangup (terminate) signal, WDHNGUP ignores the HANGUP signal, the PID file is unlinked (indicating the connection is closed), the write socket for the host is unlocked, the data file is unlinked (if the unlink flag is set) and WD exits.

WDIOTINS (Process Write Signals from INI) - When WD receives the TIOTINS write signal from INI, WDIOTINS resets the TIOTINS signal and clears the data file unlink flag indicating that the file is not to be unlinked after being written to the network.

WDEMTINS (Process Write Signals from INI) - When WD receives the TEMTINS write signal from INI, WDEMTINS resets the TEMTINS signal and sets the data file unlink flag indicating that the file is to be unlinked after being written to the network.

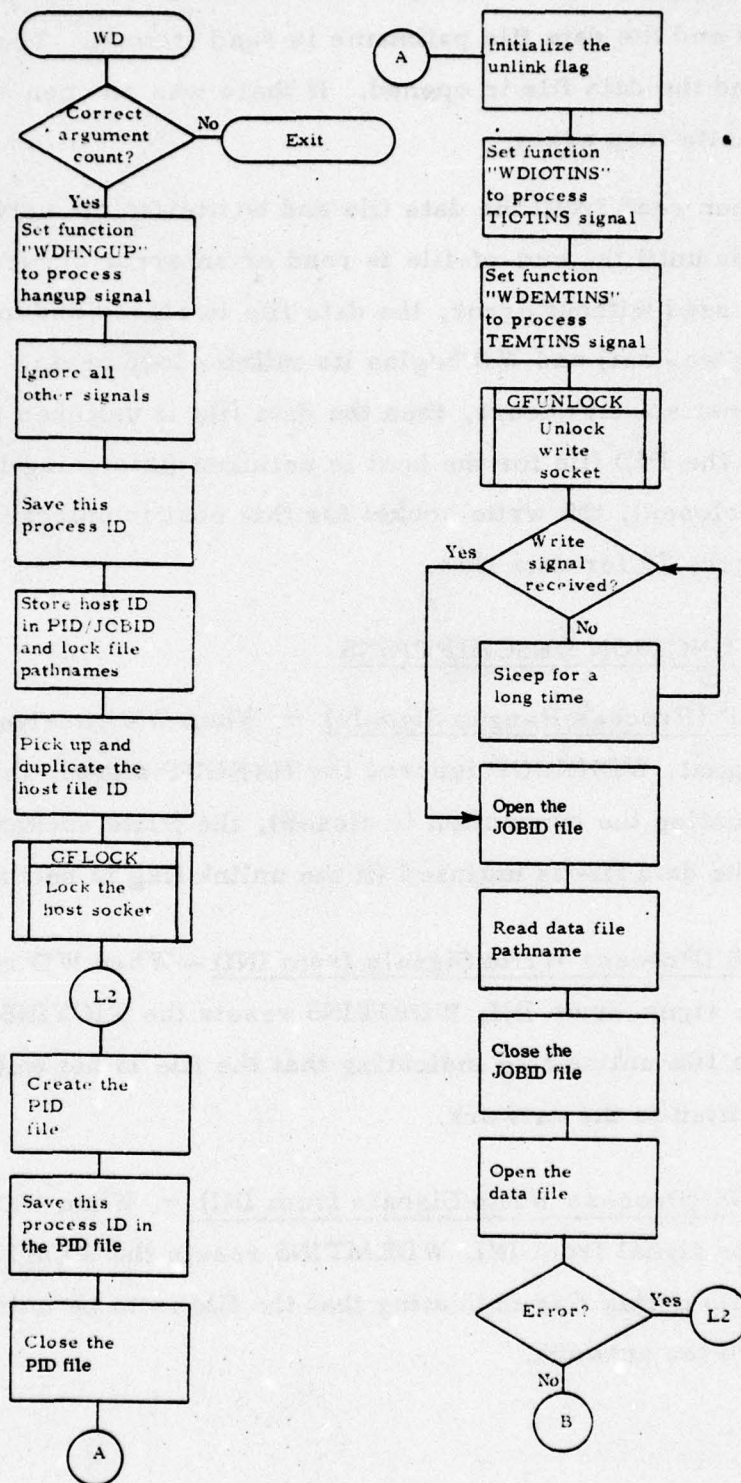


Figure 70. WD Process Data Flow (Sheet 1 of 3)

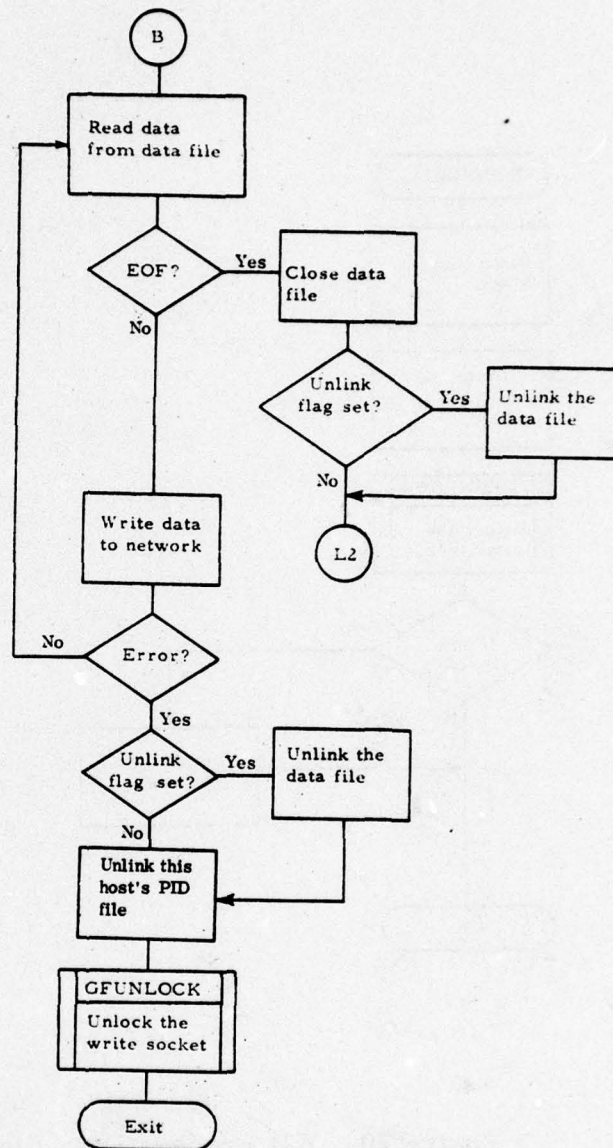


Figure 70. WD Process Data Flow (Sheet 2 of 3)

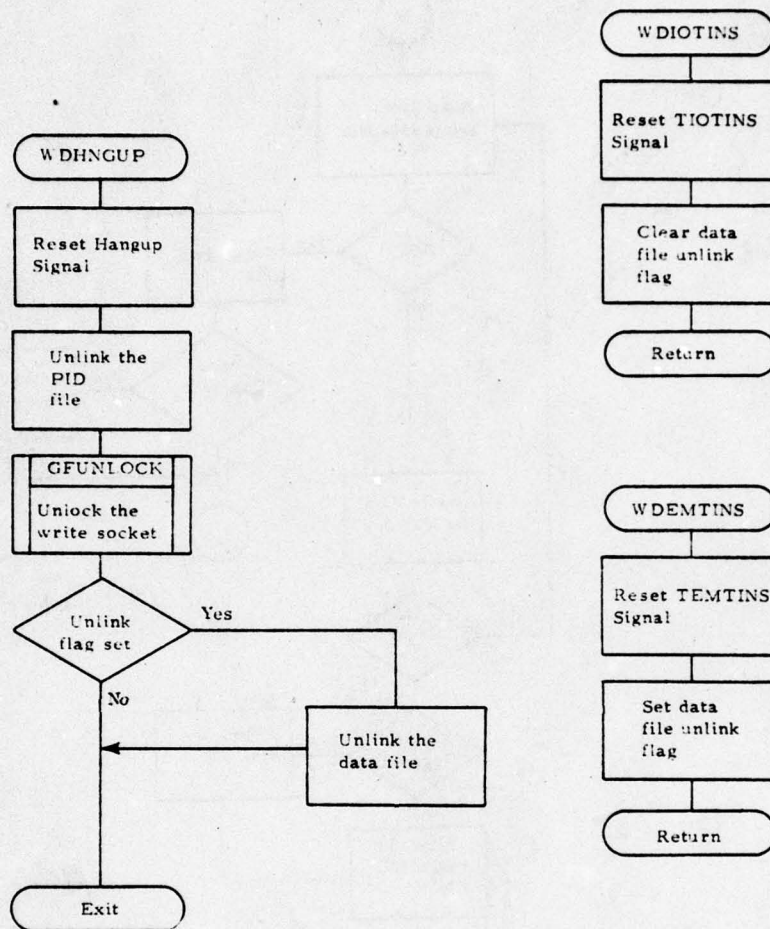


Figure 70. WD Process Data Flow (Sheet 3 of 3)

TAS GLOBAL FUNCTIONS AND DATA

This section provides descriptions of the global functions and global data implemented for TAS. Global functions and data are defined as those functions and data which are commonly utilized by the many processes of TAS. Functions are defined in the literal sense as in the C language, hence are not processes but are actual functions which are compiled with the individual processes requiring their services. Global data are different in this respect since only a single copy of these data exists in the TAS environment. These data are referred to as global because they are usually utilized by many of the TAS processes.

This section is divided into two subsections, one describing global functions and the other providing descriptions of the many global data structures and arrays existing in TAS. Six global functions have been defined for TAS and thirty-six global data structures/arrays are defined for TAS.

Global Functions

TAS requires the use of six global functions:

1. GFERROR - outputs general diagnostic messages.
2. GFLOCK - locks "read-for-update" data files.
3. GFUNLOCK - unlocks "read-for-update" data files.
4. GFJOBID - returns a unique TAS job number.
5. GFMOVE - moves 'n' sequential bytes.
6. GFNEQUAL - compares 'n' sequential bytes.

Following are the descriptions of these functions.

GFERROR -

Input Parameters - GFERROR requires two input parameters:

- a. A TAS error message number (integer).
- b. A parameter pointing to a null-terminated character string or equal to zero.

Output Parameters - None

Operation - GFERROR provides TAS with a common means for the output of diagnostic and system error messages to the user. The first input parameter specifies a canned message and the second parameter provides a character string which is to be inserted into an appropriate place in the canned message, if applicable. If the message is not a system error, the error message number indicates the appropriate error description item in GFEDES (error description file) which, in turn, contains pointers to the canned message stored in GFEMES. The character string pointed to by the second input parameter (if non-zero), is stored in the message and output to the user's terminal. GFERROR then returns to the calling function. If the message is a system error, the TAS operator's console is locked via GFLOCK and the message (as stored in GFSYSERR) is output to the console. Once the message has been printed, the operator's console is unlocked via GFUNLOCK. The user is then informed that a system error has occurred while his request was processed and GFERROR returns to the calling function.

GFLOCK -

Input Parameters - GFLOCK requires one input parameter: a path-name specifying a global data file.

Output Parameters - None

Operation - For certain TAS global data, it is necessary to protect their integrity during update sequences; i. e., for "read-for-update" protection on a given file. That is, only one process may "read-for-update" a given file at any one time. The TAS global function GFLOCK (and its counterpart, GFUNLOCK) provides this protection within the TAS environment.

GFLOCK uses the input parameter, a global data file pathname, to "lock out" the file during a "read-for-update" sequence. The character string "-lock" is appended to the file pathname to generate the lock-file pathname. GFLOCK then attempts to create the lock-file with the "read-only" mode. If this file already exists, an error status is received and the function sleeps for one second and then attempts the create again. Eventually, the create is successful and GFLOCK returns to the calling function. At this point, the specified global data file is "locked out" from other "read-for-update" attempts. Of course, this assumes the cooperation of other processes to call GFLOCK prior to attempting to "read-for-update" a file. The file can now be read in, updated, and then written back out. After the updating process is completed, it is mandatory that GFUNLOCK be called to unlock the global data file.

GFUNLOCK -

Input Parameters - GFUNLOCK requires one input parameter: a path-name specifying a global data file.

Output Parameters - None.

Operation - GFUNLOCK is a counterpart to GFLOCK and must be called when a given locked global file has been updated and written out. Basically, GFUNLOCK appends the character string "-lock" to the file pathname and unlinks the associated lock-file thus allowing another process to create it, hence gaining control of the file. Of course, failure to call GFUNLOCK after successfully locking out the file with GFLOCK would eventually bring the TAS system to a halt.

GFJOBID -

Input Parameter - GFJOBID has one input parameter: the address of a six-character buffer.

Output Parameter - GFJOBID returns the status of zero if the process was successful or an error status if a system error occurred, i.e., read or write error.

Operation - When called, GFJOBID first saves the interrupt signal state and ignores further interrupts. GJOBID is then opened, locked (via GFLOCK) and read. The contents of GJOBID are copied into the user buffer pointed to by the input argument. The JOBID counter is then incremented and rewritten. GJOBID is then closed and unlocked (via GFUNLOCK). The interrupt signal status is reset and GFJOBID returns with the status of any system errors which may have occurred during processing or a zero if none occurred.

GFMOVE -

Input Parameters - GFMOVE has three input parameters:

- 1) The number of sequential bytes to be moved.
- 2) Pointer to the buffer from which the characters are moved.
- 3) Pointer to the buffer to which the characters are moved.

Output Parameter - GFMOVE returns a pointer to the end of the destination buffer.

Operation - When called, GFMOVE copies the number of sequential bytes given by its first argument from the buffer pointed to by its second argument to the buffer pointed to by its third argument.

GFNEQUAL -

Input Parameters - GFNEQUAL has three input parameters: the first parameter is the number of sequential bytes to be compared while the second and third parameters are pointers to the two buffers to be compared.

Output Parameters - A value of one is returned if the specified number of bytes are equal in both buffers, zero otherwise.

Operation - GFNEQUAL tests for the equality of two buffer areas, pointed to by its second and third arguments, respectively, over the number of sequential bytes given by its first argument.

TAS GLOBAL DATA

The following structure and array (a la C-language) descriptions constitute a complete set of TAS system data descriptions. For each data structure, if applicable, an item drawing is provided showing all defined fields indicating whether they are single byte, double byte, or multiple byte fields. Also, a description for each field is provided. The global data descriptions are placed in alphabetical order for easy reference. The following global data are described:

- a. GACCM - access mode codes
- b. GAMAP - access mode bit maps
- c. GAPNM - application names
- d. GAPPL - application authorization data
- e. GAUTH - user authorization data
- f. GBQFN - user batch query filenames
- g. GBQST - user batch query statuses
- h. GCLAS - classification codes
- i. GDOWN - TASDOWN lock file
- j. GFEDES - error message descriptions
- k. GFEMES - error message text
- l. GFILE - user text file descriptions
- m. GFILS - file authorization data
- n. GHOST - host statuses

o.	GILOG	- interactive application logon information
p.	GIMSG	- INI batch message descriptions
q.	GJOBID	- JOBID counter
r.	GJOBS	- BQRD batch job descriptions
s.	GLBUF	- COINI record buffer
t.	GLJOBS	- COINI record descriptions
u.	GMAP	- classification bit maps
v.	GMASK	- classification bit masks
w.	GMESG	- user message descriptions
x.	GOPARAMS	- network open parameter packet
y.	GPIDS	- daemon process IDs
z.	GPROJ	- project descriptions
aa.	GQFIL	- batch query filenames for authorization check
bb.	GRESP	- batch query response processor pathnames
cc.	GTAUT	- terminal authorization data
dd.	GTCTS	- network message traffic counts
ee.	GTERM	- terminal descriptions
ff.	GUDES	- user descriptions
gg.	GUTY	- user type descriptions
hh.	LTPID	- logging tape daemon process ID
ii.	LTSIZE	- number of current logging tape records
jj.	TLTAL	- TAS activity log

GACCM (Access Mode Codes) - This two-dimensional character array contains access modes which qualify users' access authorization capabilities for applications' files. This file is parallel to GAMAP. The following is the actual C-language definition for GACCM:

```
char *GACCM[ ]{"RETRIEVE", "UPDATE", 0};
```

GAMAP (Access Mode Bit Maps) - This one-dimensional array contains the bit maps for the access modes which qualify users' access capabilities for applications' files. GAMAP is parallel to GACCM. The following is the actual C-language definition for GAMAP:

```
char GAMAP [ ] {1, 3};
```

GAPNM (Application Names) - This two-dimensional character array contains the names of the TAS and of the applications which it can access. GAPNM is parallel to the Host Descriptions (GHOST). Both GAPNM and GHOST are accessed by a host (application) ID. The actual C-language definition is given below:

```
char *GAPNM [ ] {"TAS", "RYETIP", "DIAOLS", "SOLIS",  
"ISSPIC", 0};
```

GAPPL (Application Authorization Data) - This structure contains one item for each application which:

- a. a user is authorized to access within his current project,
- b. a terminal is authorized to access.

The number of items in the user's GAPPL is given by item GNPAP of his GPROJ file; for a terminal, it is given by GNTAP within its GTAUT file. Figure 71 is an item description.

WORD 0

GANAM

GANFL

Figure 71. GAPPL Item Description

Field Descriptions

GANAM - Host ID which indexes into GAPNM and GHOST.

GANFL - Number of files accessible under this application (number of items in GFILS).

GAUTH (User Authorization Data) - This file contains administrative and access authorization data for the user. It is created/maintained by the AAFM process and is used by the Access Authorization Process (AAP) and other processes. Figure 72 is an item description.

Field Descriptions

GUPID - Index into the project descriptions (GPROJ) for the user's current project. This number converted to ASCII and appended to 'proj' yields the user's current project directory name.

GUCLR - User security clearance which is stored as a bit map.

GNPRJ - Current number of project description items in GPROJ for this user.

GNUCP - Number of compartment names specified for this user.

GUTYP - Code designator for user type.

0 = student user type

1 = operational user type

2 = crisis user type

Word 0	GUPID	GUCLR
Word 1	GNPRJ	GNUCP
Word 2	GUTYP	(not used)
Word 3	GUSNI	
Word 4		
Word 5		
Word 6	GUSSN	
Word 7		
Word 8		
Word 9		
Word 10	GUNAM	
Word 11		
Word 12		
Word 26	GUADR	
Word 27		
Word 76	GUTEL	
Word 77		
Word 86	GUORG	
Word 87		
Word 91		
Word 92	GUCMP	

Figure 72. GAUTH Item Description

- GUSNI - Contains eight character structured network identifier for this user.
- GUSSN - Contains the users social-security number in up to ten characters.
- GUNAM - Thirty character user name.
- GUADR - One hundred character user address.
- GUTEL - Contains twenty character user telephone number.
- GUORG - Ten character user agency organization designator.
- GUCMP - Variable length (size given by GNUCP) list of six character compartment names for this user.

GBQFN (User Batch Query Filenames) - TAS maintains a GBQFN file for each user. This structure contains one item for each batch host file referenced by the user in a specified batch query. The entries into GBQFN are made by the Batch Query Processors (BQPs). The filenames referenced in the query are collected into GQFIL and then copied into the next available GBQFN item(s) (pointed to by GFAUTH of GBQFN item zero, which is otherwise not used). A pointer to the first GBQFN item for a specific job is saved in the GBQST item (GFPTR) for that job. GBQFN items are deleted by LOGOFF, CANCEL, REMOVE, RECLAIM and PURGE. Figure 73 is an item description.

Field Descriptions

- GFLNM - Ten character filename
- GFAM - Single character access mode: Retrieve (1), Update (2)
- GFAUTH - Pointer to next available item (for item zero). Number of filename items for this query if first item for a specified job; otherwise, null.

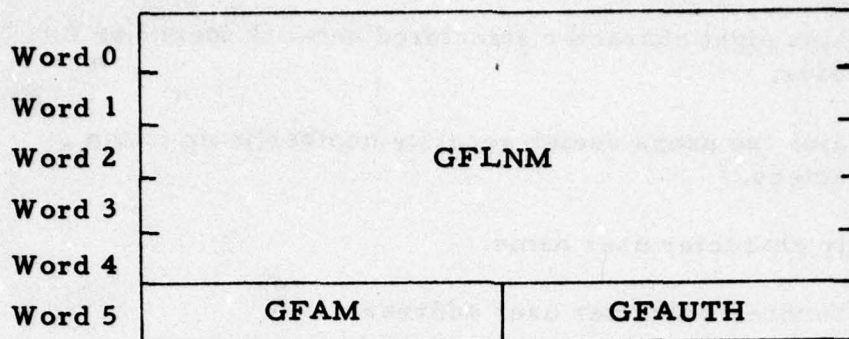


Figure 73. GBQFN Item Description

GBQST (User Batch Query Statuses) - This structure contains the descriptions of a user's current batch query. Item zero contains the number of current batch queries. Figure 74 is an item description.

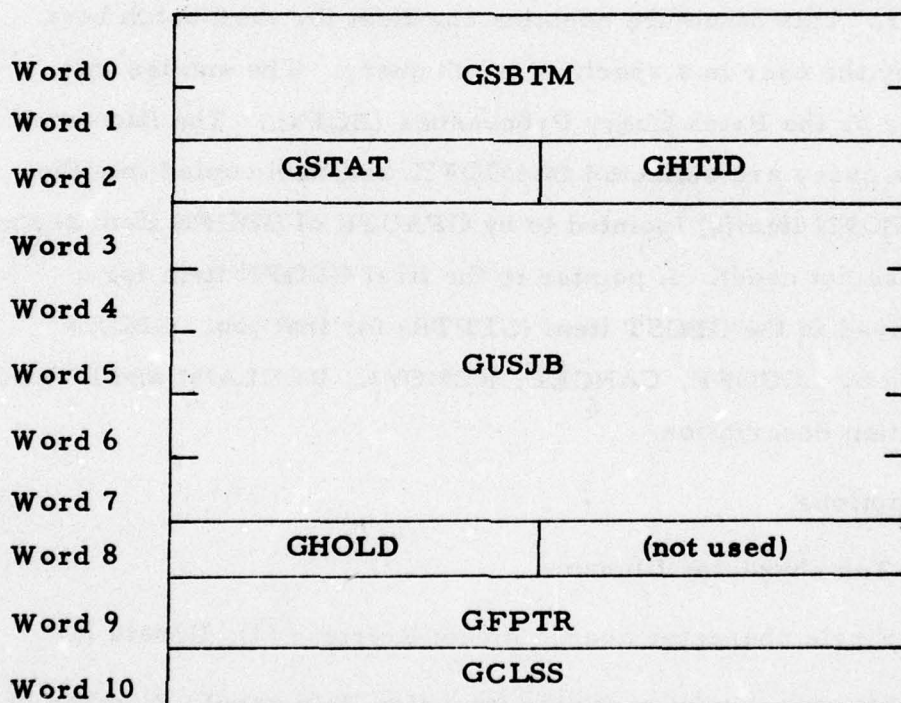


Figure 74. GBQST Item Description

Field Descriptions

GSBTM - Date Time Group (DTG) of submission of this query, or of response to this query.

GSTAT - Query status:

0 = Pending

1 = Answer received

2 = Abort(FAULT) received

3 = Abort(CAUSE) received

4 = Message logged out

5 = Lost

Note: negative values (-1 through -5) indicate delivery.

GHTID - Host ID of the application to which this query was sent.

GUSJB - Ten character JOBID for this query.

GHOLD - Hold flag: 0 = no, 1 = yes.

GFPTR - Pointer to a list of filenames associated with this query in GBQFN.

GCLSS - Classification bit-map applicable for GSTAT = 1 or -1.

GCLAS (Classification Codes) - GCLAS is a two-dimensional array which contains the classification codes assigned to responses from batch queries. The array is parallel to GMAP and GMASK which contain the classification access capability bit-maps and classification bit-masks. The actual C-language definition of GCLAS is:

```
char *GCLAS ('K', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 0);
```

The classification codes correspond to clearances as shown in figure 75.

Code	Index	Classification
K	0	Unclassified
A	1	Top Secret (SI)
B	2	Top Secret (TK)
C	3	Top Secret (SI/TK)
D	4	Top Secret
E	5	Secret (SI)
F	6	Secret (TK)
G	7	Secret (SI/TK)
H	8	Secret
I	9	Confidential (SI)
J	10	Confidential

Figure 75. Classification Codes

GDOWN (TASDOWN Lock File) – GDOWN is a file operated by the TASDOWN process which serves to inform all processes that TAS is being brought down and no new users may log on, thus effectively locking TAS.

This file is removed by a command in the "/etc/rc" TAS initialization file when TAS is booted up.

GFEDES (Error Message Descriptions) – GFEDES is a structure containing pertinent information concerning messages stored in character message buffer GFEMES. This file is built by ERRINIT and used in conjunction with GFEMES by global function GFERROR to output instructional or diagnostic messages. Figure 76 is an item description.

Word 0	GFEPTR	
Word 1	GFECHAR	GFNAME

Figure 76. GFEDES Item Description

Field Descriptions

GFEPTR - Pointer to the first character of the message stored in GFEMES.

GFECHAR - Number of characters in the message.

GFNAME - Relative pointer to the start of the inserted data if such information is pertinent; otherwise, it is a minus one.

GFEMES (Error Message Text) - GFEMES is a character array containing actual TAS messages which, when used in conjunction with GFEDES, is the mechanism by which GFERROR outputs instructional and diagnostic messages.

GFILE (User Text File Descriptions) - This structure contains the descriptions of a user's files. Each item of the structure describes a file. Item zero contains a count of the number of files the user currently has. Figure 77 is an item description.

Field Descriptions

GFNAM - File name, terminated by a null.

GDCRE - Date Time Group (DTG) of this file's creation.

AD-A050 967

LOGICON INC SAN DIEGO CALIF
TERMINAL ACCESS SYSTEM (TAS) SYSTEM DESIGN SPECIFICATION.(U)
JAN 78 R M BARNHART, L R ERICKSON
76-C-0899-6

F/G 9/2

N00014-76-C-0899
NL

UNCLASSIFIED

7 OF 8
AD
A050 967



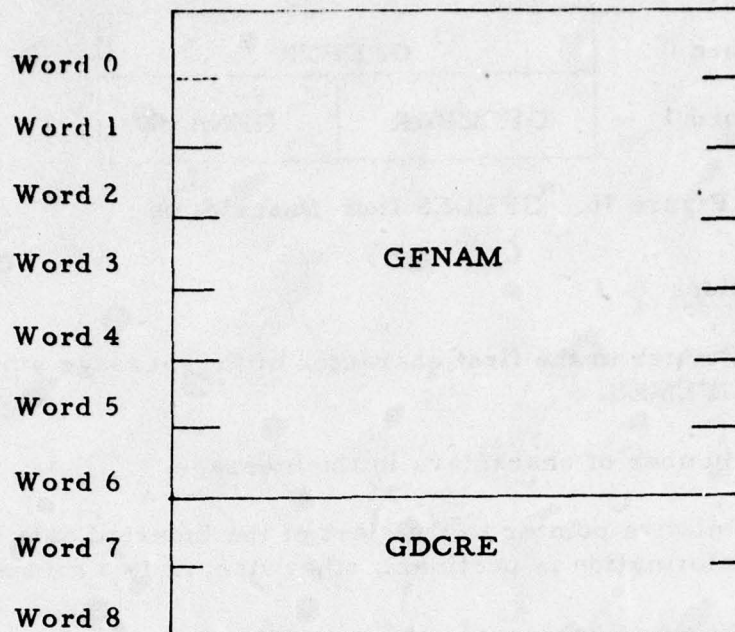


Figure 77. GFILE Item Description

GFILS (File Authorization Data) - This structure contains one item for each batch file which:

- a. a user may access under his current project/application,
- b. a terminal may access for a specified application.

The number of items in a user's GFILS is given by item GANFL of his GAPPL file; for a terminal it is given by the corresponding item in its GAPPL file. Figure 78 is an item description.

Field Descriptions

GFLNM - Ten character file name.

GFLAC - File access mode authorization.

0 = retrieve access

1 = update access

Word 0

Word 1

Word 2

Word 3

Word 4

Word 5

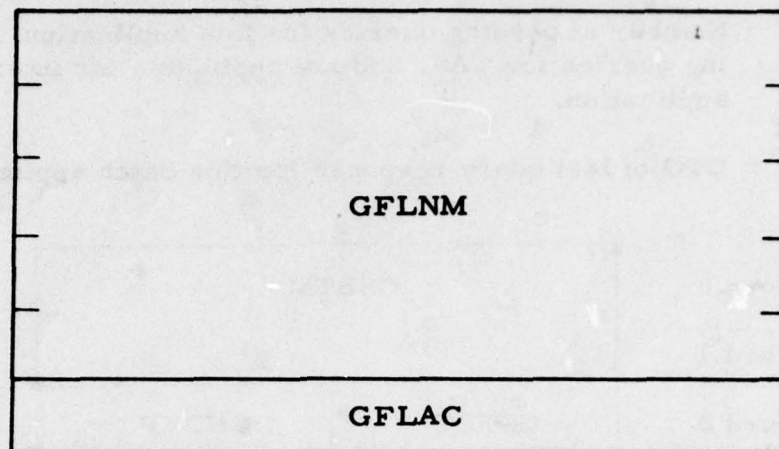


Figure 78. GFILS Item Description

GHOST (Host Statuses) - This structure contains the current status of the hosts on the COINS II network. Each item describes an application system. GHOST is indexed by a host ID which is determined by a name match in the Application Names (GAPNM), to which it is parallel. Figure 79 is an item description.

Field Descriptions

GSBTM - Date Time Group (DTG) of submission of query whose response time is in GRSTM (Batch application only).

GSTAT - Host status (Batch/interactive):

0 = READY/up

1 = BREAK/down

2 = ALARM/unknown

GHTYP - Host type:

0 = Own Self

1 = Batch

2 = Interactive

GNMPQ - Number of pending queries for this application. Total pending queries for TAS, and not applicable for interactive application.

GRSTM - DTG of last query response for this batch application.

Word 0	GSBTM	
Word 1		
Word 2	GSTAT	GHTYP
Word 3	GNMPQ	
Word 4	GRSTM	
Word 5		

Figure 79. GHOST Item Description

GILOG (Interactive Application Logon Information) - This structure contains the logon information required for interactive applications. The data are stored as a two-dimensional array of three 30-character arrays. Since each interactive application requires different logon information in different formats, the GILOG is designed to accommodate the various data. For each application, the order of the logon information in GILOG is given in the AAFM language definition.

GIMSG (INI Batch Message Descriptions) - GIMSG is a structure used by INI to store information concerning batch messages sent across the COINS II network. The message's data are deleted when a RCPT is received by INI. GIMSG entries are used to store and retrieve INI message data files in the INI message directory. Figure 80 is an item description.

Word 0	INTIM	
Word 1		
Word 2	INJOB	
Word 3		
Word 4		
Word 5		
Word 6	INSEG	
Word 7	INFCN	INTRM
Word 8		
Word 9	IHOST	

Figure 80. GIMSG Item Description

Field Descriptions

- INTIM - Time of last transmission.
- INJOB - Ten character JOBID.
- INSEG - Segment count for the message.
- INFCN - Type of job:
- 1 = INTG
 - 2 = SRVC (PRINT)
 - 3 = SRVC (TRACE)
 - 4 = SRVC (CHECK)
 - 5 = SRVC (READY)
 - 6 = SRVC (BREAK)

INTRM -- Number of transmissions.

IHOST -- Host ID (which indexes into GAPNM and HOST) of the application to which the job was sent.

GJOBID (JOBID Counter) -- GJOBID contains the next available TAS JOBID number for all TAS processes requiring a JOBID. The command process JOBID is used to display the current value of GJOBID or to set it to some desired value. Within a TAS process, the function GFJOBID may be used to obtain the current value of GJOBID and increment the JOBID counter.

The counter consists of six ASCII digits, left padded with zeroes and terminated with a null.

GJOBS (BQRD Batch Job Descriptions) -- This structure contains descriptions of all pending jobs for a specified batch host. Each item describes a job. Figure 81 is an item description.

Field Descriptions

GSBTM -- Date Time Group (DTG) of this job's submission.

GSTAT -- Job status. Values are:

0 = Pending

1 = Transmitted

2 = RLSE received

3 = Cancelled

GFUNC -- Function of this job:

1 = INTG

2 = SRVC (PRINT)

3 = SRVC (TRACE)

4 = SRVC (CHECK)

5 = SRVC (READY)

6 = SRVC (BREAK)

GRPTR -- Pointer into array GRESP to a pathname of a process to be called when a response is received for this job.

GJBID -- Ten character JOBID for this job.

GJBTR -- Ten character JOBID being traced if GFUNC = 3.

Word 0	GSBTM	
Word 1		
Word 2	GSTAT	GFUNC
Word 3	GRPTR	
Word 4		
Word 5	GJBID	
Word 6		
Word 7		
Word 8		
Word 9		
Word 10	GJBTR	
Word 11		
Word 12		
Word 13		

Figure 81. GJOBS Item Description

GLBUF (COINI Record Buffer) - GLBUF is a buffer, maintained by the TAS Logging Processes, for the COINI record logging tape. As COINI records are created and completed ready for transmission to CNMMS they are buffered into GLBUF. When ten COINI records have been accumulated, they are copied onto magnetic tape as a single physical tape record. The file is then recreated to zero length, ready for the next ten records.

The file is organized into items the length of a COINI record, or 144 bytes each. Item zero is an index which contains the number of COINI records in the file and a list of the JOBIDs for those records. Items 1 - 10 contain the actual COINI records and are the only portion of the file which is copied onto magnetic tape. Figure 82 is the GLBUF description.

Field Descriptions

NRECS - Word zero of GLBUF contains the number of COINI records currently in GLBUF.

JOBID - Words 1 - 50 contain up to ten, ten character JOBIDs of the COINI records contained in GLBUF.

COINI - Beginning at word 72 are up to ten 144 byte COINI records.

GLJOBS (COINI Record Descriptions) - This structure contains one item for each COINI record currently being processed by the TAS Logging Processes. Before creating the COINI record disk file, TLP enters the JOBID of the record into GLJOBS. Entries are also deleted by TLP when the COINI logging records are copied onto magnetic tape and then deleted. Figure 83 is an item description.

Report 76-C-0899-6

Word 0	NRECS
Word 1	
:	JOBID
Word 5	
	:
Word 46	
:	JOBID
Word 50	
Word 51	
:	(not used)
Word 71	
Word 72	
:	COINI RECORD 1
Word 143	
:	:
Word 720	
:	COINI RECORD 10
Word 791	

Figure 82. GLBUF Description

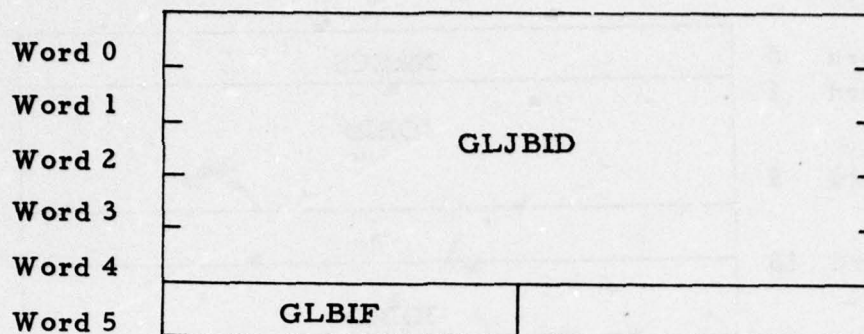


Figure 83. GLJOBS Item Description

Field Descriptions

GLJBID - Ten character JOBID for batch COINI records or interactive session ID for interactive records.

GLBIF - Flag for batch record (0) or interactive record (1).

GMAP (Classification Bit Maps) - GMAP is a numeric array which contains the bit maps for the eleven classifications of clearances for the TAS environment. GMAP is parallel to GCLAS and GMASK. User and terminal clearances are assigned via AAFM. The assigned clearance is matched to a clearance in GCLAS, which indexes into GMAP yielding the bit map to which the user/terminal clearance is set. Each bit set in the clearance indicates the type of sensitive, classified material the user or terminal has authorization to view. Figure 84 lists the various clearances with the associated bit-maps.

The C-language definition for GMAP is:

```
char GMAP [ ] 10,          /* 0 - Unclassified */
                0267,       /* 1 - Top Secret (SI) */
                0155,       /* 2 - Top Secret (TK) */
```

Bit Position	Octal Value	Classification
0	1	Confidential
1	2	Confidential (SI)
2	4	Secret
3	010	Secret (TK)
4	020	Secret (SI)
5	040	Top Secret
6	0100	Top Secret (TK)
7	0200	Top Secret (SI)

Figure 84. Classification Bit Maps

```

0377,      /* 3 - Top Secret (SI/TK) */
045,       /* 4 - Top Secret */
027,       /* 5 - Secret (SI) */
015,       /* 6 - Secret (TK) */
037,       /* 7 - Secret (SI/TK) */
5,         /* 8 - Secret */
3,         /* 9 - Confidential (SI) */
11;        /*10 - Confidential */

```

i. e., the bit map for Secret (SI) is octal 027 which means bits 0, 1, 2, and 4 are set, implying the authorization to view Unclassified, Confidential, Confidential (SI), Secret and Secret (SI) classified material.

GMASK (Classification Bit Masks) – GMASK is a numeric array which contains the bit masks for the eleven classifications of clearances for

the TAS environment. GMASK is parallel to GCLAS and GMAP. The following is an example of the use of the three arrays in cross reference.

Given:

- 1) User's clearance is a bit map (GMAP) which is stored in GAUTH . GUCLR and
- 2) A batch response clearance maps onto a clearance in GCLAS with index of x.

Then a user is cleared to see the response if

$GMASK(x) = GUDES.GUCLR \& GMASK(x)$

is true, given that "&" is a logical 'and' operation.

The C-language definition for GMASK is:

```
char GMASK [ ] {0,          /* 0 - Unclassified */
                0200,        /* 1 - Top Secret (SI) */
                0100,        /* 2 - Top Secret (TK) */
                0300,        /* 3 - Top Secret (SI/TK) */
                040,         /* 4 - Top Secret */
                020,         /* 5 - Secret (SI) */
                010,         /* 6 - Secret (TK) */
                030,         /* 7 - Secret (SI/TK) */
                4,           /* 8 - Secret */
                2,           /* 9 - Confidential (SI) */
                1 };        /*10 - Confidential */
```

Figures 84 and 85 display the association between clearance and bit position.

Clearance (See GMAP)	GMASK	Clearance & GMASK	Is GMASK equal to (Clearance & GMASK)?
0377	0200	0200	yes
0267	0300	0200	no
0155	0100	0100	yes
045	0200	0	no
037	020	020	yes
027	030	020	no
015	020	010	no
5	2	020	yes

Figure 85. Determination of Access Authorization

GMESG (User Message Descriptions) - This structure contains the descriptions of a user's current messages. Each item of the structure describes a message. Item zero contains the number of messages. Figure 86 is an item description.

Field Descriptions

GSBTM - Date Time Group (DTG) of origination of this message

GORIG - User ID of the originator of this message

Word 0	GSBTM
Word 1	
Word 2	GORIG

Figure 86. GMESG Item Description

GOPARAMS (Network Open Parameter Packet) - GOPARAMS is a structure which is the mechanism for TAs to specify exactly how the NCP connection is to be opened for a given host. The open call to the NCP specifies the address of this packet which contains information stating exactly how the connection is to be opened by the NCP. Figure 87 is an item description.

Field Descriptions

- | | |
|----------|--|
| GOP | - Opcode for kernel and daemon which is unused. |
| GTYPE | - Connection type bit string (not set/set). |
| | 01 = ICP/direct |
| | 02 = user/server |
| | 04 = listen/init |
| | 010 = general/specific |
| | 020 = simplex/duplex |
| | 040 = absolute/relative |
| GID | - ID of the file for the kernel and daemon which is unused here. |
| GLSKT | - Local socket number which is either absolute or relative. |
| GFSKT | - Foreign socket number which is either absolute or relative. |
| GFRNHOST | - Foreign host number for a connection to a specific host number |
| | 0102 = RYETIP |
| | 05 = DIAOLS |
| | 02 = SOLIS |
| | 03 = ISSPIC |

- GBSIZE - The byte size for the connection - TELNET demands 8.
- GNOMALL - The nominal allocation for the connection. This is used for opens for reads.
- GTIMEO - The number of seconds to wait before timing out.
- GRELID - The FID of the file on which the data connection is based.

Word 0	GOP	GTYPE
Word 1	GID	
Word 2	GLSKT	
Word 3	GFSKT	
Word 4		
Word 5	GFRNHOST	GBSIZE
Word 6	GNOMALL	
Word 7	GTIMEO	
Word 8	GRELID	

Figure 87. GOPARAMS Item Description

GPIDS (Daemon Process IDs) - GPIDS is a sequential file containing, in one word items, the process IDs of some TAS daemons. Each TAS daemon which must be externally killed writes its process ID at the end of the file. The command process TASDOWN uses GPIDS to terminate those daemons which have placed their process IDs in GPIDS.

GPROJ (Project Descriptions) - This structure contains one item for each project under which the user is authorized to work. The number of items in the structure is given by item GNPPJ of the user's GAUTH file. GPROJ is indexed by a project ID which can be determined by an item-by-item search. Item GUPID of GAUTH is the project ID for the user's current project; this ID converted to ASCII and appended to 'proj' gives the user's current project directory. Item zero is for the default project. Figure 88 is an item description.

Field Descriptions

GPJID - Ten character project name

GNPAP - Number of applications defined and authorized for this user
(number of items in GAPPL)

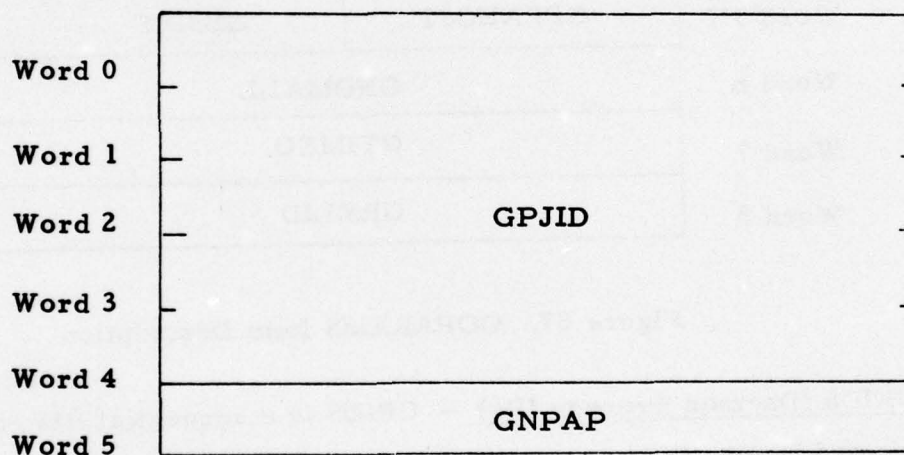


Figure 88. GPROJ Item Description

GQFIL (Batch Query Filename for Authorization Check) – This structure contains one item for each filename referenced by a user in a batch query. GQFIL is created by the Batch Query Processors (BQPs) to pass the file-names queried by the user to AAP for authorization validation and to TLP for logging purposes. GQFIL is deleted by the BQP which created it when the user's query becomes a valid TAS job.

The number of items in GQFIL (equal to the number of files referenced in the query) is not specified but may be counted by reading GQFIL items until end-of-file.

For purposes of buffer allocation, the parameter TMXFPQ defines the maximum number of file references allowed per batch query. Figure 89 is an item description.

Field Descriptions

GFLNM – Ten character filename

GFAM – Single character file access mode:

1 = Retrieve

2 = Update

GFAUTH – Single character access authorization code for this file (returned by AAP):

0 = User/terminal authorized access to file

1 = User not authorized access to file

2 = Terminal not authorized access to file

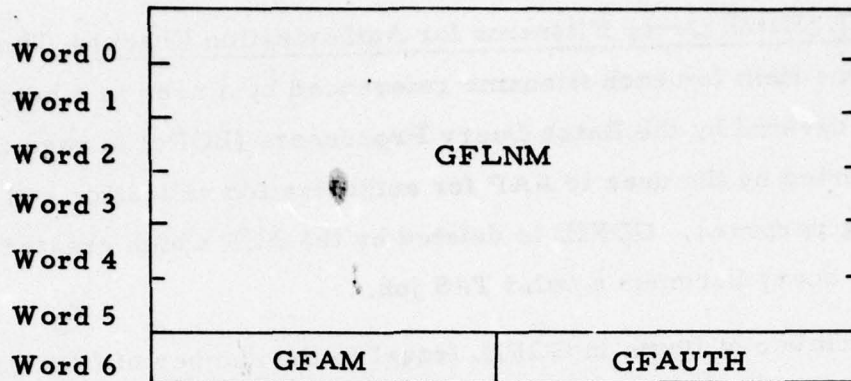


Figure 89. GQFIL Item Description

GRES P (Batch Query Response Processor Pathnames) - This character array contains the pathnames of the processes to be called when job responses are received. GRESP is indexed by pointers stored in the Jobs Description file (GJOBS). GRESP is used and maintained by the Batch Query and Response Dispatcher (BQRD).

GTAUT (Terminal Authorization Data) - This structure contains fixed administrative and authorization data for a TAS terminal. Figure 90 is an item description.

Field Descriptions

GTNAP - Number of applications authorized to this terminal; also, number of items in the terminal's GAPPL file.

GTCLR - Terminal clearance bit-map.

GTOWN - Ten character name of terminal owner.

GTTID - One character terminal identifier.

GTNCP - Number of compartment names specified for this terminal

GTLOC - One hundred character terminal location descriptor

GTTEL - Twenty character terminal telephone number

GTCMP - Variable length (size depending on GTNCP) list of six character compartment names for this terminal.

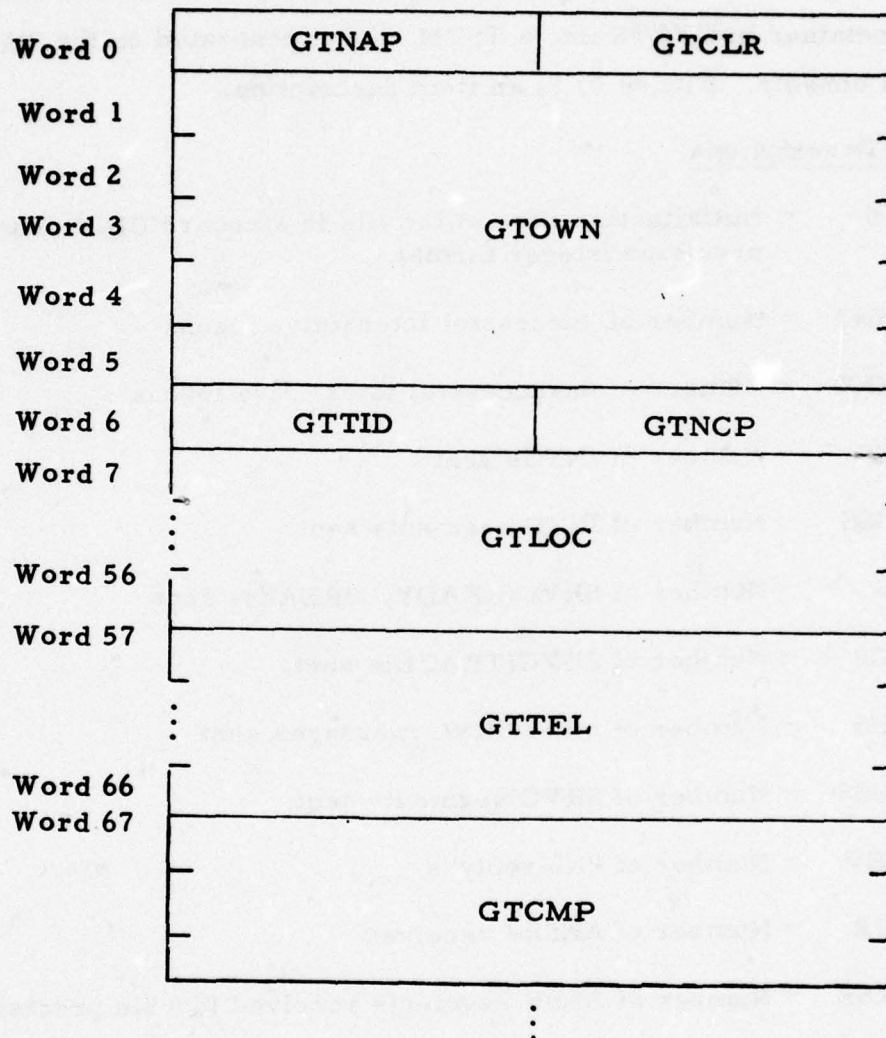


Figure 90. GTAUT Item Description

GTCTS (Network Message Traffic Counts) – This structure is an array of integer counters for the significant message types sent and received by TAS. The file is created and initialized by the TSUM process at TAS system initialization time. The IQI and INI processes update these message counters during TAS operation and the TSUM process is responsible for formatting the data contained in GTCTS into a TSUM report generated on the TAS operator's console. Figure 91 is an item description.

Field Descriptions

TTIME	– Initialization time of the file in standard UNIX double precision integer format.
TSUCINT	– Number of successful interactive logons
TUNSINT	– Number of unsuccessful interactive logons
TINTGS	– Number of INTGs sent
TINTGSS	– Number of INTG segments sent
TSRBS	– Number of SRVC(READY, BREAK)s sent
TSTRCS	– Number of SRVC(TRACE)s sent
TSOTHS	– Number of other SRVC messages sent
TSRVCSS	– Number of SRVC segments sent
TNOREP	– Number of "No reply"s
TANSRR	– Number of ANSRs received
TANSRSR	– Number of ANSR segments received (double precision)
TRLSER	– Number of RLSEs received
TABFR	– Number of ABRT(FAULT)s received

Word 0	TTIME
Word 1	
Word 2	TSUCINT
Word 3	TUNSINT
Word 4	TINTGS
Word 5	TINTGSS
Word 6	TSRBS
Word 7	TSTRCS
Word 8	TSOTHS
Word 9	TSRVCSS
Word 10	TNOREP
Word 11	TANSRR
Word 12	TANSRSR
Word 13	
Word 14	TRLSER
Word 15	TABFR
Word 16	TABCR
Word 17	TABER
Word 18	TABRTSR
Word 19	TSRBR
Word 20	TSTRKR
Word 21	TSOTHR
Word 22	TSRVCSR

Figure 91. GTCTS Item Description

TABCR - Number of ABRT(CAUSE)s received
 TABER - Number of ABRT(ERROR)s received
 TABRTSR - Number of ABRT segments received
 TSRBR - Number of SRVC(READY, BREAK)s received
 TSTRKR - Number of SRVC(TRACK)s received
 TSOTHR - Number of other SRVC messages received
 TSRVCSR - Number of SRVC segments received

GTERM (Terminal Descriptions) - This structure is used to map a TAS terminal ID onto a UNIX terminal ID. The index into GTERM is the TAS station number and the contents of that character is the character to be appended to 'tty' to create the UNIX terminal ID. Figure 92 is an item description.

Field Descriptions

GCHAR - last character of 'ttyx' for this terminal
 GTLOG - terminal status

0 = terminal logged off
 1 = user logged onto this terminal
 2 = terminal being updated
 3 = terminal being created or deleted

GUDES (User Descriptions) - This structure contains one item for each TAS user. Each item contains information and current status for the user and is indexed by a user ID. The user ID can be determined by an item by item search on the user's SNI, or a program called directly or indirectly by a logged on user can get the user ID by using the system call 'GETUID'. Figure 93 is an item description.

Word 0

GCHAR

GTLOG

Figure 92. GTERM Item Description

Field Descriptions

- GPJID** - Ten character project name under which the user is working if the user is logged on.
- GSNI** - Eight character structured network identifier (SNI).
- GAMOU** - Numeric designator for user classification:
- 0 = ordinary user
 - 1 = administrative user
 - 2 = master user
- GLOGG** - User status code:
- 0 = logged off
 - 1 = logged on
 - 2 = being updated
 - 3 = being created/deleted
- GLINK** - Pointer to another agency-group member in an administrative user/sibling link list.
- GUTTY** - User's UNIX terminal identifier if the user is currently logged on.
- GUTID** - User's terminal station number (index into GTERM) if the user is logged onto TAS.

Word 0	GPJID	
Word 1		
Word 2		
Word 3		
Word 4		
Word 5	GSNI	
Word 6		
Word 7		
Word 8		
Word 9	GAMOU	GLOGG
Word 10	GLINK	
Word 11	GUTTY	GUTID

Figure 93. GUDS Item Description

GUTY (User Type Descriptions) – This two dimensional character array contains the three user type classifications into which users are grouped. The actual C-language definition for GUTY is as follows:

```
char *GUTY [ ] { "STUDENT", "OPER", "CRISIS" };
```

LTPID (Logging Tape Daemon Process ID) – LTPID is created by the Logging Tape Daemon (LTD) and contains the one word process ID of LTD. TLP uses this file to obtain the process ID in order to send LTD a signal to write the contents of GLBUF onto magnetic tape. In similar fashion, the LOGTAPE command process uses LTPID to send LTD a signal to mount

a new logging tape. The TASDOWN process also uses LTPID to obtain LTD's process ID in order to terminate it.

LTSIZE (Number of Current Logging Tape Records) – LTSIZE is created and maintained by the Logging Tape Daemon (LTD). This file contains the number of logging tape records currently written on the tape in the form of six ASCII digits (padded on the left with zeroes) and terminated by a new line character.

TLTAL (TAS Activity Log) – The TAS Activity Log (TLTAL or TAL) is a sequential file containing records of varying format and length which describe significant TAS events. These records are written into the TAL by the TAS Logging Processes (TLP) and are queried for formatting and output by the TALIST command. Termination of the current TAL (optionally saving the file in another designated file) is handled by the TALEND command.

There are currently fourteen events processed by TLP and the following thirteen of these cause entries to be made into the TAL:

<u>Event Code</u>	<u>Event Description</u>
00	TAS logon
01	TAS logoff
02	TAS user submits a batch query
03	RCPT received for batch query
04	RLSE received for batch query
05	Response received for batch query
07	Batch query display attempt
08	Batch query cancelled

<u>Event Code</u>	<u>Event Description</u>
09	SRVC(TRACE) returns "job lost"
10	Interactive host logon attempt
11	Interactive host logoff
12	Unauthorized application attempt
13	Unauthorized batch host file access attempt

Each record begins with a header block containing the following fields.
Figure 94 is a header description.

Field Descriptions

GEVENT - The code for this event (00 - 13).

GTALSZ - The number of bytes in the event description record, not including the six bytes of the header.

GEVTIM - The time of this event in standard UNIX double precision integer format.

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		

Figure 94. TLTAL Record Header Description

The following paragraphs provide a description of the content and format of the event description record for each of the thirteen events recorded in the TAL.

Note: Because the event records often contain similar data in varying positions within those records, it is not possible to define global C structures with fixed names for all the variables. Because these record structures are defined local to the function in which they are used, all variables referenced in the following descriptions which do not begin with 'G' are preceded with several characters ("TL" for TLP and "TA_" for TALIST) in order to form their actual name.

Event 00 (TAS logon attempt) - When a user attempts (perhaps unsuccessfully) to log onto TAS, TLP records the following information in the TAL.

Figure 95 is an event 00 record description.

Field Descriptions

- SNI** - Eight character Structured Network Identifier of the user attempting the logon.
- TERMID** - Last character of the terminal at which the logon attempt was made.
- LOGONF** - Flag for successful (0) or unsuccessful (1) logon attempt.

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		
Word 3		
Word 4	SNI	
Word 5		
Word 6		
Word 7	TERMID	LOGONF

Figure 95. Description of Event 00 Record

Event 01 (TAS logoff) - When a user logs off TAS, TLP records the following information in the TAL. See figure 96 for an event 01 record description.

Field Description

SNI - Eight character Structured Network Identifier of the user logging off.

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		
Word 3	SNI	
Word 4		
Word 5		
Word 6		

Figure 96. Description of an Event 01 Record

Event 02 (User submits a batch query) - When a TAS user submits a batch query and the query passes access authorization, and grammatical and syntactic validation becoming a TAS job, TLP records the following information. See figure 97 for an event 02 record description.

Field Description

- JOBID - Ten character JOBID for this query.
- GASNI - Eight character Structured Network Identifier of the user submitting the query.
- GAPROJ - Ten character name of the user's current project under which the query was submitted.

GATERM - Last character of the terminal name at which the query was submitted (i. e., tty'2').

APPL - Application ID of the queried host.

NFILES - The number of batch host files referenced in this query. This references the number of GQFIL like items which immediately follow.

The following data fields form a GQFIL like structure which follows the record in the form of one GQFIL item for each batch host file referenced.

GFNNM - Ten character file name.

GFAM - File access mode (1=Retrieve, 2=Update).

GFAUTH - File access authorization (0=user/terminal authorized, 1=user not authorized, 2=terminal not authorized, 3=neither authorized).

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		
Word 3		
Word 4		
Word 5	JOBID	
Word 6		
Word 7		
Word 8		
Word 9	GASNI	
Word 10		
Word 11		
Word 12		
Word 13	GAPROJ	
Word 14		
Word 15		
Word 16		
Word 17	GATERM	
Word 18	APPL	NFILES
Word 19	GFLNM	
Word 20		
Word 21		
Word 22		
Word 23		
Word 24	GFAM	GFAUTH

⋮

Figure 97. Description of an Event 02 Record

Event 03 (RCPT received for batch query) - When a RCPT message is received from a distant batch host for a query job sent by TAS, TLP records the following information in the TAL. See figure 98 for an event 03 record description.

Field Descriptions

JOBID - Ten character TAS JOBID for this query.

RCPTF - Flag for receipt received (0) or receipt not received (1).

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		
Word 3	JOBID	
Word 4		
Word 5		
Word 6		
Word 7		
Word 8	RCPTF	

Figure 98. Description of an Event 03 Record

Event 04 (RLSE received for batch query) - When a RLSE message is received from a distant batch host for a query job sent by TAS, TLP records the following information in the TAL. See figure 99 for an event 04 record description.

Field Descriptions

JOBID - Ten character TAS JOBID for this query.
NETID - Ten character network ID for this query.
HOST - Host ID.

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		
Word 3		
Word 4		
Word 5	JOBID	
Word 6		
Word 7		
Word 8		
Word 9	NETID	
Word 10		
Word 11		
Word 12		
Word 13	HOST	

Figure 99. Description of an Event 04 Record

Event 05 (Response received for batch query) - When TAS receives a response for a batch query, TLP records the following information in the TAL. Figure 100 is an event 05 record description.

Field Descriptions

JOBID - Ten character TAS JOBID for this query.

NETID - Ten character network ID for this query.

RESP - Response type:

0 = No Response

1 = ANSR

2 = ABORT CAUSE

3 = ABORT HOST

4 = ABORT ERROR

5 = ABORT FAULT

CLASS - Two character response classification code.

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		
Word 3	JOBID	
Word 4		
Word 5		
Word 6		
Word 7	NETID	
Word 8		
Word 9		
Word 10		
Word 11		
Word 12	CLASS	
Word 13		
Word 14		

Figure 100. Description of an Event 05 Record

Event 07 (Batch query display attempt) - When a user attempts to display the response to a batch query, TLP records the following information in the TAL. Figure 101 is an event or record description.

Field Descriptions

- | | |
|--------|--|
| JOBID | - Ten character TAS JOBID for this query. |
| GASNI | - Eight character Structured Network Identifier of the user attempting the display. |
| GAPROJ | - Current project name (10 characters) of the user under which the display attempt was made. |
| GATERM | - Last character of the terminal at which the display attempt was made, (e. g. , tty'2'). |
| CLASS | - Two character response classification code. |
| AUTH | - User-terminal response access authorization code. |
| | 00 = User-terminal authorized to view the response. |
| | 01 = User not authorized to view the response. |
| | 02 = Terminal not authorized to view the response. |
| | 03 = Neither authorized to view response. |

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		
Word 3	JOBID	
Word 4		
Word 5		
Word 6		
Word 7		
Word 8	GASNI	
Word 9		
Word 10		
Word 11		
Word 12	GAPROJ	
Word 13		
Word 14		
Word 15		
Word 16		
Word 17	GATERM	
Word 18	CLASS	
Word 19	AUTH	

Figure 101. Description of an Event 07 Record

Event 08 (Batch query cancelled) - When a batch query is cancelled by a user or by TAS, TLP records the following information in the TAL. Figure 102 is an event 08 record description.

Field Description

- JOBID - Ten character JOBID of this query.
- SNI - Eight character Structured Network Identifier of the user submitting the query.
- QSTAT - Query status at time of query logoff:
 00 = INTG not sent to network
 01 = No RCPT, no reply
 02 = RCPT, no reply
 03 = Response received

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		
Word 3		
Word 4		
Word 5	JOBID	
Word 6		
Word 7		
Word 8		
Word 9	SNI	
Word 10		
Word 11		
Word 12		
	QSTAT	

Figure 102. Description of an Event 08 Record

Event 09 (Unknown Job ID from network) - When TAS receives a SRVC (TRACK) from a distant host in response to a user-generated SRVC (TRACE) indicating that a batch query job has been lost, TLP records the following information in the TAL. Figure 103 is an event 09 record description

Field Descriptions

JOBID - Ten character JOBID for this query.

HOST - Host ID.

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		
Word 3	JOBID	
Word 4		
Word 5		
Word 6		
Word 7	HOST	
Word 8		

Figure 103. Description of an Event 09 Record

Event 10 (Interactive host logon attempt) - When a user attempts to initiate an interactive session, TLP records the following information. Figure 104 is an event 10 record description.

Field Descriptions

- JOBID** - Ten character interactive session ID.
- GASNI** - Eight character Structured Network Identifier of the user attempting the interactive session.
- GAPROJ** - Ten character name of the current project under which the interactive host logon attempt was made.
- GATERM** - Last character of the terminal name at which the interactive host logon attempt was made.
- APPL** - Host ID.
- LOGONF** - Flag for successful logon (0) or unsuccessful logon (1).

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		
Word 3		
Word 4		
Word 5	JOBID	
Word 6		
Word 7		
Word 8		
Word 9	GASNI	
Word 10		
Word 11		
Word 12		
Word 13		
Word 14	GAPROJ	
Word 15		
Word 16		
Word 17	GATERM	
Word 18	APPL	LOGONF

Figure 104. Description of an Event 10 Record

Event 11 (Interactive session logoff) - When the user logs off an interactive host or the interactive session is terminated, TLP records the following information in the TAL. Figure 105 is an event 11 record description.

Field Descriptions

JOBID	- Ten character interactive session ID.
GASNI	- Eight character Structured Network Identifier of the user logging off.
GAPROJ	- Ten character name of the project under which the user was conducting the interactive session.
GATERM	- Last character of the terminal name at which the user was conducting the interactive session.
HOST	- Host ID
TERM_CODE	- Termination code for the interactive session: 0 = successful 1 = no reply 2 = timed-out 3 = too many retransmissions 4 = unable to send/receive from network 5 = addressee (host) down 6 = session aborted by addressee

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		
Word 3	JOBID	
Word 4		
Word 5		
Word 6		
Word 7	GASNI	
Word 8		
Word 9		
Word 10		
Word 11	GAPROJ	
Word 12		
Word 13		
Word 14		
Word 15		
Word 16	GATERM	
Word 17		
Word 18	HOST	TERM_CODE

Figure 105. Description of an Event 11 Record

Event 12 (Unauthorized Application access attempt) - When a user makes an unauthorized attempt to access a TAS host, TLP records the following information in the TAL. Figure 106 is an event 12 record description.

Field Descriptions

- GASNI** - Eight character Structured Network Identifier of the user attempting illegal access.
- GAPROJ** - Ten character name of the current project under which the user attempted the access.
- GATERM** - Last character of the terminal name at which the access was made.
- HOST** - Host ID.
- HAC** - Host access code:
 00 = User-terminal authorized access to application.
 01 = User not authorized access to application.
 02 = Terminal not authorized access to application.
 03 = Neither authorized access to application.

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2	GASNI	
Word 3		
Word 4		
Word 5		
Word 6	GAPROJ	
Word 7		
Word 8		
Word 9		
Word 10	GATERM	
Word 11		
Word 12	HOST	HAC
Word 13		

Figure 106. Description of an Event 12 Record

Event 13 (Unauthorized Batch host file access attempt) - When a user attempts to query a distant batch host file to which either he or his terminal is not authorized access, TLP records the following information in the TAL. Figure 107 is an event 13 record description.

Field Descriptions

- GASNI - Eight character Structured Network Identifier of the user attempting the access.
- GAPROJ - Ten character current project name, under which the user attempted the batch host file access.
- GATERM - Last character of the terminal name at which the user attempted the access.
- HOST - Host ID.
- AUTH - Access authorization code for the batch host file(s):
00 = User-terminal authorized access to all batch host files queried.
01 = User not authorized access to some batch host files queried.
02 = Terminal not authorized access to some batch host files queried.
03 = Neither authorized to some batch host files queried.
- NFILES - Number of files queried. Also refers to the number of GQFIL-like items following the record.

The following data fields form a GQFIL-like item for each file referenced in the query and immediately follow the above data in the record.

- GFLNM - Ten character file name.
- GFAM - File access mode (0=Retrieve, 1=Update).
- GFAUTH - Authorization code for user-terminal access attempt:
00 = User-terminal authorized access to this batch host file.
01 = User not authorized access to this batch host file.

02 = Terminal not authorized access to this batch host file.

03 = Neither authorized access to this batch host file.

Word 0	GEVENT	GTALSZ
Word 1	GEVTIM	
Word 2		
Word 3	GASNI	
Word 4		
Word 5		
Word 6		
Word 7	GAPROJ	
Word 8		
Word 9		
Word 10		
Word 11		
Word 12	GATERM	
Word 13	HOST	AUTH
Word 14	NFILES	
Word 15	GFLNM	
Word 16		
Word 17		
Word 18		
Word 19		
Word 20	GFAM	GFAUTH

⋮

Figure 107. Description of an Event 13 Record

APPENDIX A

ACCESS AUTHORIZATION FILE MAINTENANCE LANGUAGE

Introduction

The Access Authorization File Maintenance Language, AAFM, provides a flexible, user-oriented language enabling full file maintenance for Access Authorization (AA) files. AAFM, available to the TASMASTER and those users designated as administrative users, is the facility for creating, modifying, and deleting AA entries. In addition to the AA file maintenance facility, AAFM has display capabilities which permit a privileged user either an overview or a closeup of either user or terminal files.

AAFM is available to privileged users requesting the AAFM process as a TAS application. The TASMASTER's AA files are created when the TAS COINS II Environment is initially generated. The TASMASTER's AA files can be updated but never deleted. The TASMASTER creates, maintains, and deletes all administrative user's AA files, any ordinary user's AA files, and all terminals' AA files.

An administrative user's AA files contain those access rights that have been assigned to the agency-group represented by that administrative user. The administrative user, in turn, has the ability to create, delete, or modify ordinary user's authorization files for the given agency-group. He is able to assign an ordinary user any authorization which was assigned to him when his sni was created. The administrative user and those users that he maintains have the same first five digits in their snis (sni-prefix). There is at most one administrative user for any given sni-prefix.

The ordinary user is an individual who may use the TAS environment in the capacity he is authorized for. He is not authorized to use the AAFM language.

Two basic types of AA files exist: user files and terminal files. Terminal authorization files are created and maintained by the TASTMASTER for each user terminal connected to TAS. The terminal AA files identify the terminal (e.g., location, terminal number) and the applications and per-application files the terminal may access. The terminal AA files directly correspond to individual user AA files with the exception of the concept of projects.

The AAFM language, conceptually, self-divides into natural blocks. The most apparent division is, of course, between the maintenance of either user or terminal AA files. The concept of maintenance itself is further subdivided into the creating, modifying, listing, and deleting of AA files. These divisions directly result in the structure of the "main" AAFM commands, which are:

- a. CHANGE sni-name
- b. CREATE sni-name
- c. DELETE sni-name
- d. LIST sni-name
- e. CHANGE TERM termid
- f. CREATE TERM termid
- g. DELETE TERM termid
- h. LIST TERM termid

where sni-name is an sni and termid is a terminal identifier.

The remaining conceptual compartmentalization of AAFM is best presented by means of an outline of the necessary user and terminal access authorization information. The outline follows.

User Information

a. Basic Information

1. SNI identification
2. Name
3. Social Security Number
4. Clearance
5. Administrative, Master, or Ordinary user
6. Crisis, operational, or student user type
7. Agency organization
8. Full agency address
9. Telephone number
10. Compartment list

b. Project Information

1. Project Name (or none for default project)
2. Application Information
 - a) Interactive Application Information
 - 1) Application Name
 - 2) Interactive Application logon information
 - b) Batch Application Information
 - 1) Application Name
 - 2) File Information

- a File name
- b File access

Terminal Information

a. Basic Information

1. Terminal identification
2. Clearance
3. Location
4. Telephone number
5. Agency Organization
6. Compartment list

b. Application Information

1. Batch Application Information
 - a) Application name
 - b) File information
 - 1) File name
 - 2) File access
2. Interactive Application Name

The formal definitions which follow are divided into eleven basic groups.

1. Basic Definitions
2. Create Sni Command Block
3. Change Sni Command Block
4. Delete Sni Statement
5. Create Terminal Command Block
6. Change Terminal Command Block

7. Delete Terminal Statement
8. List Sni Statement
9. List Terminal Statement
10. Execute Statement
11. Quit

Where the AAFM processor encounters syntactic or semantic errors, informative error messages are printed out enabling the AAFM user to input a correct version of the erroneous statement.

BASIC DEFINITIONS

The fundamental definitions for the AAFM language consist of the character set, integer, character constant and keywords. These syntactic definitions are referenced throughout the remainder of the AAFM language definitions.

CHARACTER SET

The American Standard Code for Information Interchange (ASCII) is the character set used for AAFM. Out of this character set, the following delimiters, digits, and letters are valid for AAFM statement construction.

Syntax:

```

delimiter = '|Δ|,|;
digit      = 0|1|2|3|4|5|6|7|8|9
letter     = A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|
             S|T|U|V|W|X|Y|Z|a|b|c|d|e|f|g|h|i|j|k|
             l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

```


Semantics: The meaning of each of the delimiters is as follows:

- ' (single quotation mark) delimits (initial and terminal) character constants
- Δ designates a space
- , (comma) delimits items in a list
- ; terminates AAFM statements

CHARACTER-CONSTANT AND EXPRESSION

Two character structures are used to allow the input of information with unpredictable formats (such as an address): character-constant-list and character-constant. A character-constant consists of a series of letters, digits, or other printable characters enclosed by single quotation marks. Character-constant is the major structure to the input data in AAFM. The valid maximum length of a character-constant depends on the situation. The character-constant-list is a series of character-constants separated by commas.

Syntax:

```

< character-constant-list > = < character-constant > , < character-
                                constant-list > | < character-constant >
< character-constant >      = '< characters >'
< characters >              = < letter > < characters > | < digit >
                                < characters > | < delimiter > < characters > |
                                < letter > | < digit > | < delimiter >
    
```

INTEGER

The integer is used in AAFM to express TAS terminal station numbers, which are either one- or two-digit integers.

Syntax:

<integer> = <digit> <integer> | <digit>

KEYWORDS

Keywords are words with an inherent meaning and use within the AAFM language. Figure A-1 includes the keywords used in AAFM. They are generally grouped by function.

KEYWORD	MEANING	AREA OF CONCERN
ADD		
CHANGE		
CREATE		
DELETE		
EXECUTE		
LIST		
QUIT		
ADDRESS		
APP	Application	
APP-B	Batch Application	
APP-I	Interactive Application	
CLEAR	Clearance	
COMP	Compartment	
END		
EPROJ	End Project	Project Definition

Figure A-1. AAFM Keywords (Sheet 1 of 3)

KEYWORD	MEANING	AREA OF CONCERN
FILE		
LOCATION		
NAME		
ORG	Agency/Organization	
PASSWORD		
PROJECT		
SNI		
SSN	Social Security Number	
TELE	Telephone	
TERM	Terminal	
UTYPE	User Type	
R	Retrieve	File Access Mode
U	Update	File Access Mode
OPER	Operational	User Type (UTYPE)
CRISIS	Crisis	User Type (UTYPE)
STUDENT	Student	User Type (UTYPE)
SOLIS	SOLIS Host	Interactive Application
DIAOLS	DIAOLS Host	Batch Application
ISSPIC	ISSPIC Host	Batch Application
RYETIP	RYETIP Host	Batch Application
AD	Administrative User	AAFM Capabilities
A	Top Secret SI	Clearance
B	Top Secret TK	Clearance

Figure A-1. AAFM Keywords (Sheet 2 of 3)

KEYWORD	MEANING	AREA OF CONCERN
C	Top Secret SI/TK	Clearance
D	Top Secret	Clearance
E	Secret SI	Clearance
F	Secret TK	Clearance
G	Secret SI/TK	Clearance
H	Secret	Clearance
I	Confidential SI	Clearance
J	Confidential	Clearance
K	Unclassified	Clearance

Figure A-1. AAFM Keywords (Sheet 3 of 3)

CREATE SNI COMMAND BLOCK

The creation of a user's AA files is accomplished by a series of statements, collectively referred to as a create-sni-cmd-blk. The create-sni-cmd-blk contains four to five basic parts. The create-sni-stat initiates the block while the basic personnel information block (basic-sni-block) follows and contains necessary data concerning the user, his clearance, and his organization. The third constituent is the default project block. This block contains the basic user access authorization record. If the user has any projects assigned to him, they are defined after the default project definition. The specific applications and specific files which he can access when working within the constraints of the project are input. The last statement of the create-sni-cmd-blk is the end statement which terminates the user's AA file definition.

Syntax:

`<create-sni-cmd-blk> = <create-sni-stat> <create-sni-blk>
 <end-stat>`

`<create-sni-stat> = CREATE <sni-name> AD ; |
 CREATE <sni-name> ;`

`<sni-name> = <character-constant>`

`<create-sni-blk> = <basic-sni-blk> <add-sni-app-blk>
 <add-proj-blk> | <basic-sni-blk>
 <add-proj-blk>`

`<basic-sni-blk> = <basic-sni-stat> <basic-sni-blk> |
 <basic-sni-stat>`

`<basic-sni-stat> = <name-stat> | <clear-stat> | <ssn-stat> |
 <password-stat> | <utype-stat> | <org-stat> |
 <sni-stat> | <address-stat> | <tele-stat> |
 <add-comp-stat> | |`

`<name-stat> = NAME <character-constant> ;`

`<clear-stat> = CLEAR <clear-code> ;`

`<clear-code> = A | B | C | D | E | F | G | H | I | J | K`

`<ssn-stat> = SSN <character-constant> ;`

`<password-stat> = PASSWORD <character-constant> ;`

`<utype-stat> = UTYPE <utype-code> ;`

`<utype-code> = CRISIS | OPER | STUDENT`

`<org-stat> = ORG <character-constant> ;`

<address-stat> = ADDRESS <character-constant>;
 <tele-stat> = TELE <character-constant>;
 <sni-stat> = SNI <sni-name>;
 <add-comp-stat> = <comp-clause>; | ADD <comp-clause>;
 <comp-clause> = COMP <character-constant-list>
 <add-proj-blk> = <add-proj-section> <add-proj-blk> |
 <add-proj-section> ||
 <add-proj-section> = <add-proj-stat> <add-sni-app-blk>
 <proj-stat>
 <add-proj-stat> = <proj-clause>; | ADD <proj-clause>;
 <proj-clause> = PROJECT <proj-name>
 <proj-name> = <character-constant>
 <add-sni-app-blk> = <add-app-sni-section> <add-sni-app-blk> |
 <add-app-sni-section>
 <add-app-sni-section> = <add-app-B-section> | <add-app-I-sni-stat>
 <add-app-B-section> = <add-app-B-stat> <add-file-blk>
 <add-app-B-stat> = <add-B-phrase>; | ADD <app-B-phrase>;
 <app-B-phrase> = APP-B <app-B-name>
 <app-B-name> = DIAOLS | ISSPIC | RYETIP
 <add-app-I-sni-stat> = <app-I-sni-clause>; | ADD <app-I-sni-
 clause>;
 <app-I-sni-clause> = <app-I-phrase> <app-I-info-phrase>

<app-I-phrase>	= APP-I <app-I-name>
<app-I-name>	= SOLIS
<app-I-info-phrase>	= < character-constant-list >
<add-file-blk>	= < add-file-stat > <add-file-blk> < add-file-stat >
<add-file-stat>	= < file-clause > ; ADD <file-clause> ;
<file-clause>	= < file-phrase> < access-code > < file-phrase>
<access-code>	= R U
<file-phrase>	= FILE <character-constant>
<eproj-stat>	= EPROJ;
<end-stat>	= END;

CHANGE SNI COMMAND BLOCK

A TAS user's access authorization files are modified via the change sni command block. The user's administrative user and the TASTMASTER are the only authorized individuals who may alter the user's files. As in the case of the creation of a user, at no time may an ordinary user have access to information which his administrative user does not have.

There exist five basic parts of the change and command block.

1. The change-sni statement initiates the block.
2. The change-basic-sni-blk is a collection of statements which update personnel, clearance, and organization information.
3. The default-project modifications are input next.

4. Additional projects are added, changed, or deleted.

5. The end statement terminates the command block.

Syntax:

```

<change-sni-cmd-blk>  = <change-sni-stat> <change-sni-blk>
                        <end-stat>

<change-sni-stat>     = CHANGE <sni-name> ;

<change-sni-blk>      = <change-basic-sni-blk> <proj-blk>|
                        <change-basic-sni-blk> <change-sni-
                        app-blk> <proj-blk>

<change-basic-sni-blk> = <change-basic-sni-stat> <change-basic-
                        sni-blk> | <change-basic-sni-stat>

<change-basic-sni-stat> = <basic-sni-stat> | <delete-comp-stat>

<delete-comp-stat>    = DELETE <comp-clause> ;

<proj-blk>            = <project-blk> <proj-blk> | <project-blk>

<project-blk>         = <change-proj-stat> <change-sni-app-blk>
                        <eproj-stat> | <add-proj-blk> |
                        <delete-proj-stat> ||

<change-proj-stat>    = CHANGE <proj-clause> ;

<delete-proj-stat>    = DELETE <proj-clause> ;

<change-sni-app-blk>  = <change-app-sni-section> <change-sni-
                        app-blk> | <change-app-sni-section> ||

```

```

<change-app-sni-section> = <change-app-B-section> |
                           <delete-app-B-stat> |
                           <change-app-I-stat> |
                           <delete-app-I-stat>

<change-app-B-section> = <change-app-B-stat> <file-blk>

<change-app-B-stat>     = CHANGE <app-B-phrase> ;

<file-blk>              = <file-stat> <file-blk> | <file-stat>

<file-stat>             = <add-file-stat> | <change-file-stat> |
                           <delete-file-stat>

<change-file-stat>      = CHANGE <file-clause> ;

<delete-file-stat>      = DELETE <file-phrase> ;

<change-app-I-stat>     = CHANGE <app-I-sni-clause> ;

<delete-app-B-stat>     = DELETE <app-B-phrase> ;

<delete-app-I-stat>     = DELETE <app-I-phrase> ;

```

DELETE SNI STATEMENT

The files associated with a user are deleted when the appropriate user inputs the delete-sni statement. The TAS user cannot be logged on at the time his files are to be deleted and if the user is an administrative user, subordinates must be deleted first.

Syntax:

```

<delete-sni-stat>      = DELETE <sni-name> ;

```


CREATE TERMINAL COMMAND BLOCK

The creation of a terminal's access authorization files is accomplished by a series of statements collectively referred to as a create-term-cmd-blk. The create-term-cmd-blk creates the AA files only; it does not cause UNIX to recognize the terminal so that users can log on at the terminal.

The create-term-cmd-blk has four basic parts:

- a. Create terminal statement.
- b. Basic terminal block.
- c. Add terminal application block.
- d. End statement.

The create terminal statement initiates the block and includes the UNIX terminal identification. The basic terminal section contains the clearance and location information pertinent to the terminal. The add application block has direct relationship to the add-sni-app-blk in the create-sni-cmd-blk, and the end statement terminates the create-term-cmd-blk.

The TASMASTER is the only user who can define the terminal AA files.

Syntax:

```
<create-term-cmd-blk> = <create-term-stat> <create-term-blk>
                        <end-stat>
```

```
<create-term-stat>    = CREATE TERM <term-unix-id> ;
```

```
<term-unix-id>        = <character-constant>
```

```
<create-term-blk>     = <basic-term-blk> <add-term-app-blk>
```

`<basic-term-blk>` = `<basic-term-stat>` `<basic-term-blk>` |
`<basic-term-stat>` ||
`<basic-term-stat>` = `<clear-stat>` | `<location-stat>` | `<org-stat>` |
`<tele-stat>` | `<add-comp-stat>`
`<location-stat>` = LOCATION `<character-constant>` ;
`<add-term-app-blk>` = `<add-app-term-section>` `<add-term-`
`app-blk>` | `<add-app-term-section>` ||
`<add-app-term-section>` = `<add-app-B-section>` | `<add-app-I-term-`
`stat>`
`<add-app-I-term-stat>` = `<app-I-phrase>` ; | ADD `<app-I-phrase>` ;

CHANGE TERMINAL COMMAND BLOCK

The terminal AA files are updated by means of the change terminal command block. The terminal identifier, in the form of a UNIX terminal identifier or a TAS station number, must reference a previously defined terminal. The basic terminal information and access authorization may be updated. The TASMMASTER is the only user who may change terminal AA files.

Syntax:

`<change-term-cmd-blk>` = `<change-term-stat>` `<change-term-blk>`
`<end-stat>`
`<change-term-stat>` = CHANGE TERM `<term-clause>` ;
`<term-clause>` = `<term-tas-id>` | `<term-unix-id>`
`<term-tas-id>` = `<integer>`

```

<change-term-blk>      = <basic-term-blk> <change-term-app-blk>

<change-term-app-blk>  = <change-app-term-section>
                        <change-term-app-blk> |
                        <change-app-term-section> ||

<change-app-term-
section>                = <change-app-B-section> |
                        <delete-app-B-stat> |
                        <delete-app-I-stat>

```

DELETE TERMINAL STATEMENT

The TASMMASTER deletes a TAS terminal's AA files via the delete terminal statement. The terminal must be in a logged off state when the files are deleted.

Syntax:

```
<delete-term-stat>      = DELETE TERM <term-clause> ;
```

LIST SNI STATEMENT

An AAFM user can display the contents of the user files for which he is responsible. The information displayed depends upon the list-statement, the sni-options selected, and whether the user is an administrative user or the TASMMASTER. The presence of the sni-option-list does not alter which TAS user's AA file contents are displayed; it simply determines the order and type of data output.

Syntax:

```

<list-sni-stat>         = <list-sni-clause> ; |
                        <list-sni-clause> <sni-option-list> ;

```


<list-sni-clause>	= LIST LIST <sni-prefix> LIST <sni-name>
<sni-prefix>	= <character-constant>
<sni-option-list>	= <sni-option>, <sni-option-list> <sni-option>
<sni-option>	= ADDRESS APP CLEAR COMP FILE NAME ORG PROJECT SSN TELE UTYPE

LIST TERMINAL STATEMENT

The display of terminal AA files by the TASMMASTER is done by means of the list terminal statement. The complete list of TAS terminal station numbers plus their corresponding UNIX terminal identifiers and locations is produced when the list term statement is of the form of LIST TERM. If the list-term-stat includes a terminal identifier, it is in the form of either a TAS station number or a UNIX terminal identifier. If the terminal AA files exist, the displayed information is:

- a. TAS station number
- b. UNIX terminal identifier
- c. Location
- d. Clearance
- e. Organization
- f. Telephone number
- g. Compartment names
- h. Applications and per application files

The TASMMASTER can control the type of data and the order of its display by means of the term-option-list.

Syntax:

```

<list-term-stat>      = LIST TERM; | LIST TERM <term-clause> ; |
                        LIST TERM <term-option-list> ; |
                        LIST TERM <term-clause> <term-option-
                        list> ;

<term-option-list>    = <term-option> , <term-option-list> |
                        <term-option>

<term-option>         = APP | COMP | FILE | LOCATION | ORG |
                        TELE | CLEAR
  
```

EXECUTE COMMAND

The execute command allows AAFM users to input statement sequences from a predefined text file.

Syntax:

```

<execute-sni-cmd>      = EXECUTE <character-constant> ;

<execute-term-cmd>     = EXECUTE TERM <character-constant> ;
  
```

QUIT COMMAND

The quit command provides an exit from the AAFM process.

Syntax:

```

<quit-cmd>            = QUIT ;
  
```

AAFM YACC SPECIFICATIONS

YACC (Yet Another Compiler-Compiler) is utilized for front-end development in AAFM. This second section of Appendix A presents the AAFM grammar as YACC specifications. YACC accepts left associative LR(1) grammars as input and resolves any syntactic ambiguities if they occur. The output from YACC is a parser program and a set of tables representing the grammar. Not shown in the YACC specifications presented herein are the user-supplied lexical analyzer and parse tree generators.

AAFM utilizes YACC for two separate front-ends: 1) a front-end for parsing AAFM statements, used in the process AAFM, and 2) a front-end for terminal-type AAFM statements used in the process AATERM. Subsequent pages provide the YACC specifications for AAFM.

AAFM YACC Specification

```

%token ':' 0 AMNAME 20 CHARCONS 30 INTEGER 33
%token TRYETIP 1 TDIAOLS 2 TSOLIS 3 TISSPIC 4
%token TUNCL 50 TTSSI 51 TTSTK 52 TTSTT 53
%token TTSN 54 TSSI 55 TSTK 56 TSST 57 TSON 58 TCSI 59 TCON 60
%token TSTUD 70 TOPER 71 TCRSS 72
%token TRTRV 80 TUPDT 81
%token CREATE 100 AD 101 CREATERM 102 NAME 103 CLEAR 104
%token SSN 105 PASSWORD 106 UTYPE 107 ORG 108 ADDRESS 109
%token TELE 110 COMP 111 NAMELIST 112 ADD 113 ADDCOMP 114
%token ADDPROJ 115 ADDAPPB 116 ADDAPPI 117 ADDFILE 118
%token EPROJ 119 LOCATION 120 CHANGE 121 DELETE 122
%token CHNGUSER 123 CHNGPROJ 124 CHNGFILE 125 CHNGAPPB 126
%token CHNGAPPI 127 CHNGTERM 128 DELUSER 129 DELTERM 130
%token DELCOMP 131 DELPROJ 132 DELAPPI 133 DELAPPB 134
%token DELFILE 135 LIST 136 LISTERM 137 APPI 138
%token APPB 139 TERM 140 END 141 PROJECT 142 FILE 143
%token CHARLIST 144 QUIT 145 EXEC 146 EXCTERM 147
%token USERLIST 150 TERMLIST 151 APP 152 SNI 153
%token ',' 500 LEXERR 600
%%
list:
/* beginning of rules section */
/* list is the start symbol */
/* empty */
/* statement */
| list stat ','
= { psfinsh($2) ; }
| list error ','
= { psfinsh(0) ; } ;

stat:
CREATE CHARCONS
= { $$ = pstree1(CREATE,$2) ; }
| CREATE CHARCONS admincaluse
= { $$ = pstree2(CREATE,$2,$3) ; }
| CREATE TERM CHARCONS
= { $$ = pstree1(CREATERM,$3) ; }
| NAME CHARCONS
= { $$ = pstree1(NAME,$2) ; }
| CLEAR clearance
= { $$ = pstree1(CLEAR,$2) ; }
| SSN CHARCONS
= { $$ = pstree1(SSN,$2) ; }
| SNI CHARCONS
= { $$ = pstree1(SNI,$2) ; }
| PASSWORD CHARCONS
= { $$ = pstree1(PASSWORD,$2) ; }
| UTYPE usertype
= { $$ = pstree1(UTYPE,$2) ; }
| ORG CHARCONS
= { $$ = pstree1(ORG,$2) ; }
| ADDRESS CHARCONS
= { $$ = pstree1(ADDRESS,$2) ; }
| TELE CHARCONS
= { $$ = pstree1(TELE,$2) ; }
| addphrase
ADD addphrase
= { $$ = $2 ; }
| EPROJ
= { $$ = pstreeno(EPROJ) ; }
| CHANGE cngphrase
= { $$ = $2 ; }
| DELETE delphrase
= { $$ = $2 ; }

```

```

| END
|   = { $$ = pstreeno(END) ; }
| LIST
|   = { $$ = pstree0(LIST) ; }
| LIST uselist
|   = { $$ = pstree1(LIST,$2) ; }
| LIST CHARCONS
|   = { $$ = pstree1(LIST,$2) ; }
| LIST CHARCONS uselist
|   = { $$ = pstree2(LIST,$2,$3) ; }
| LIST TERM
|   = { $$ = pstree0(LISTERM) ; }
| LIST TERM trmlist
|   = { $$ = pstree1(LISTERM,$3) ; }
| LIST TERM CHARCONS
|   = { $$ = pstree1(LISTERM,$3) ; }
| LIST TERM CHARCONS trmlist
|   = { $$ = pstree2(LISTERM,$3,$4) ; }
| LIST TERM INTEGER
|   = { $$ = pstree1(LISTERM,$3) ; }
| LIST TERM INTEGER trmlist
|   = { $$ = pstree2(LISTERM,$3,$4) ; }
| EXEC CHARCONS
|   = { $$ = pstree1(EXEC,$2) ; }
| EXEC TERM CHARCONS
|   = { $$ = pstree1(EXETERM,$3) ; }
| QUIT
|   = { $$ = pstreeno(QUIT) ; } ;

adminclause:
AD
  = { $$ = pstreeno(AD) ; } ;

addphrase
COMP chrlist
  = { $$ = pstree1(ADDCOMP,$2) ; }
| PROJECT CHARCONS
  = { $$ = pstree1(ADDPROJ,$2) ; }
| APPB batchapp
  = { $$ = pstree1(ADDAPPB,$2) ; }
| APPI interapp
  = { $$ = pstree1(ADDAPPI,$2) ; }
| APPI interapp chrlist
  = { $$ = pstree2(ADDAPPI,$2,$3) ; }
| FILE CHARCONS
  = { $$ = pstree1(ADDFILE,$2) ; }
| FILE CHARCONS fillist
  = { $$ = pstree2(ADDFILE,$2,$3) ; } ;

cngphrase:
CHARCONS
  = { $$ = pstree1(CHNGUSER,$1) ; }
| PROJECT CHARCONS
  = { $$ = pstree1(CHNGPROJ,$2) ; }
| APPI interapp chrlist
  = { $$ = pstree2(CHNGAPPI,$2,$3) ; }
| APPB batchapp
  = { $$ = pstree1(CHNGAPPB,$2) ; }
| TERM CHARCONS
  = { $$ = pstree1(CHNGTERM,$2) ; }
| TERM INTEGER
  = { $$ = pstree1(CHNGTERM,$2) ; }
| FILE CHARCONS

```

Report 76-C-0899-6

```

      = { $$ = pstree1(CHNGFILE,$2) ; }
| FILE CHARCONS fillist
      = { $$ = pstree2(CHNGFILE,$2,$3) ; } ;

delphrase:
  CHARCONS
    = { $$ = pstree1(DEUSER,$1) ; }
  | COMP chrlist
    = { $$ = pstree1(DELCOMP,$2) ; }
  | TERM CHARCONS
    = { $$ = pstree1(DELTERM,$2) ; }
  | TERM INTEGER
    = { $$ = pstree1(DELTERM,$2) ; }
  | PROJECT CHARCONS
    = { $$ = pstree1(DELPROJ,$2) ; }
  | APPI interapp
    = { $$ = pstree1(DELAPPI,$2) ; }
  | APPB batchapp
    = { $$ = pstree1(DELAPPB,$2) ; }
  | FILE CHARCONS
    = { $$ = pstree1(DELFILE,$2) ; } ;

clearance:
  TUNCL
    = { $$ = pstreeno(TUNCL) ; }
  | TTSSI
    = { $$ = pstreeno(TTSSI) ; }
  | TTSTK
    = { $$ = pstreeno(TTSTK) ; }
  | TTSST
    = { $$ = pstreeno(TTSST) ; }
  | TTSON
    = { $$ = pstreeno(TTSON) ; }
  | TSSI
    = { $$ = pstreeno(TSSI) ; }
  | TSTK
    = { $$ = pstreeno(TSTK) ; }
  | TSST
    = { $$ = pstreeno(TSST) ; }
  | TSON
    = { $$ = pstreeno(TSON) ; }
  | TCSI
    = { $$ = pstreeno(TCSI) ; }
  | TCON
    = { $$ = pstreeno(TCON) ; } ;

interapp:
  TSOLIS
    = { $$ = pstreeno(TSOLIS) ; } ;

batchapp:
  TDIAOLS
    = { $$ = pstreeno(TDIAOLS) ; }
  | TRYETIP
    = { $$ = pstreeno(TRYETIP) ; }
  | TISSPIC
    = { $$ = pstreeno(TISSPIC) ; } ;

fillist:
  TRTRV
    = { $$ = pstreeno(TRTRV) ; }
  | TUPDT
    = { $$ = pstreeno(TUPDT) ; } ;

usertype:
  TSTUD

```



```

      = { $$ = pstreeno(TSTUD) ; }
| TCRSS
  = { $$ = pstreeno(TCRSS) ; }
| TOPER
  = { $$ = pstreeno(TOPER) ; } ;
chrlist:
  charslst
    = { $$ = pslstdon(CHARLIST) ; } ;
charslst:
  CHARCONS
    = { pslstnew($1) ; }
| charslst ',' CHARCONS
  = { pslstnxt($3) ; } ;
uselist:
  listpars
    = { $$ = pslstdon(USERLIST) ; } ;
listpars:
  listparam
    = { pslstnew($1) ; }
| listpars ',' listparam
  = { pslstnxt($3) ; } ;
listparam:
  NAME
    = { $$ = pstreeno(NAME) ; }
| CLEAR
  = { $$ = pstreeno(CLEAR) ; }
| ORG
  = { $$ = pstreeno(ORG) ; }
| ADDRESS
  = { $$ = pstreeno(ADDRESS) ; }
| TELE
  = { $$ = pstreeno(TELE) ; }
| SSN
  = { $$ = pstreeno(SSN) ; }
| UTYPE
  = { $$ = pstreeno(UTYPE) ; }
| COMP
  = { $$ = pstreeno(COMP) ; }
| PROJECT
  = { $$ = pstreeno(PROJECT) ; }
| APP
  = { $$ = pstreeno(APP) ; }
| FILE
  = { $$ = pstreeno(FILE) ; } ;
trmlist:
  tlstpars
    = { $$ = pslstdon(TERMLIST) ; } ;
tlstpars:
  tlstparam
    = { pslstnew($1) ; }
| tlstpars ',' tlstparam
  = { pslstnxt($3) ; } ;
tlstparam:
  ORG
    = { $$ = pstreeno(ORG) ; }
| LOCATION
  = { $$ = pstreeno(LOCATION) ; }
| CLEAR
  = { $$ = pstreeno(CLEAR) ; }
| TELE

```

Report 76-C-0899-6

```
| COMP = { $$ = pstreeno(TELE) ; }  
| APP  = { $$ = pstreeno(COMP) ; }  
| FILE = { $$ = pstreeno(APP) ; }  
| FILE = { $$ = pstreeno(FILE) ; } ;
```

AATERM YACC Specification

```

%token ' ' 0
%token AMNAME 20
%token CHARCONS 30
%token TRYETIP 1
%token TRTRV 80
%token TUNCL 50
%token TTSN 54
%token CREATE 100
%token SSN 105
%token TELE 110
%token ADDPROJ 115
%token EPROJ 119
%token CHNGUSER 123
%token CHNGAPPI 127
%token DELCOMP 131
%token DELFILE 135
%token APPB 139
%token CHARLIST 144
%token USERLIST 150
%token ' ' 500
%token LEXERR 600
%

```

INTEGER 33	TSOLIS 3	TISSPIC 4
TDIAOLS 2	TUPDT 81	
TTSSI 51	TTSTK 52	TTSST 53
TSSI 55	TSTK 56	TSST 57
AD 101	CREATERM 102	NAME 103
UTYPE 107	ORG 108	ADDRESS 109
COMP 111	NAMELIST 112	ADD 113
ADDAPPB 116	ADDAPPI 117	ADDFILE 118
LOCATION 120	CHANGE 121	DELETE 122
CHNGPROJ 124	CHNGFILE 125	CHNGAPPB 126
CHNGTERM 128	DELUSER 129	DELTERM 130
DELPROJ 132	DELAPPI 133	DELAPPB 134
LIST 136	LISTERM 137	APPI 138
TERM 140	END 141	PROJECT 142
QUIT 145	EXEC 146	FILE 143
TERMLIST 151	APP 152	EXETERM 147

/* beginning of rules section */

/* list is the start symbol */

/* empty */

```

| list stat ' ' /* statement */
  = { psfinsh($2); }
| list error ' ' /* syntax error */
  = { psfinsh(0); } ;

```

stat:

```

| CREATE TERM CHARCONS
  = { $$ = pstree1(CREATERM,$3) ; }
| CLEAR clearance
  = { $$ = pstree1(CLEAR,$2) ; }
| ORG CHARCONS
  = { $$ = pstree1(ORG,$2); }
| TELE CHARCONS
  = { $$ = pstree1(TELE,$2) ; }
| addphrase
  ADD addphrase
  = { $$ = $2 ; }
| LOCATION CHARCONS
  = { $$ = pstree1(LOCATION,$2) ; }
| CHANGE cngphrase
  = { $$ = $2 ; }
| DELETE delphrase
  = { $$ = $2 ; }
| END
  = { $$ = pstreeno(END) ; }
| LIST TERM
  = { $$ = pstree0(LISTERM) ; }
| LIST TERM trmlist
  = { $$ = pstree1(LISTERM,$3) ; }
| LIST TERM CHARCONS
  = { $$ = pstree1(LISTERM,$3) ; }
| LIST TERM CHARCONS trmlist
  = { $$ = pstree2(LISTERM,$3,$4) ; }
| LIST TERM INTEGER
  = { $$ = pstree1(LISTERM,$3) ; }
| LIST TERM INTEGER trmlist
  = { $$ = pstree2(LISTERM,$3,$4) ; }

```



```

| EXEC TERM CHARCONS
|   = ( $$ = pstree1(EXETERM,$3) ; )
| QUIT
|   = ( $$ = pstreeno(QUIT) ; ) ;
addphrase:
COMP chrlist
|   = ( $$ = pstree1(ADDCOMP,$2) ; )
| APPB batchapp
|   = ( $$ = pstree1(ADDAPPB,$2) ; )
| APPI interapp
|   = ( $$ = pstree1(ADDAPPI,$2) ; )
| FILE CHARCONS
|   = ( $$ = pstree1(ADDFILE,$2) ; )
| FILE CHARCONS fillist
|   = ( $$ = pstree2(ADDFILE,$2,$3) ; ) ;
cngphrase:
| APPB batchapp
|   = ( $$ = pstree1(CHNGAPPB,$2) ; )
| TERM CHARCONS
|   = ( $$ = pstree1(CHNGTERM,$2) ; )
| TERM INTEGER
|   = ( $$ = pstree1(CHNGTERM,$2) ; )
| FILE CHARCONS
|   = ( $$ = pstree1(CHNGFILE,$2) ; )
| FILE CHARCONS fillist
|   = ( $$ = pstree2(CHNGFILE,$2,$3) ; ) ;
delphrase:
| COMP chrlist
|   = ( $$ = pstree1(DELCOMP,$2) ; )
| TERM CHARCONS
|   = ( $$ = pstree1(DELTERM,$2) ; )
| TERM INTEGER
|   = ( $$ = pstree1(DELTERM,$2) ; )
| APPI interapp
|   = ( $$ = pstree1(DELAPPI,$2) ; )
| APPB batchapp
|   = ( $$ = pstree1(DELAPPB,$2) ; )
| FILE CHARCONS
|   = ( $$ = pstree1(DELFILE,$2) ; ) ;
interapp:
TSOLIS
|   = ( $$ = pstreeno(TSOLIS) ; ) ;
batchapp:
TDIAOLS
|   = ( $$ = pstreeno(TDIAOLS) ; )
| TRYETIP
|   = ( $$ = pstreeno(TRYETIP) ; )
| TISSPIC
|   = ( $$ = pstreeno(TISSPIC) ; ) ;
fillist:
TRTRV
|   = ( $$ = pstreeno(TRTRV) ; )
| TUPDT
|   = ( $$ = pstreeno(TUPDT) ; ) ;
chrlist:
charslist
|   = ( $$ = ps1stdon(CHARLIST) ; ) ;
charslist:
CHARCONS
|   = ( ps1stnew($1) ; )

```

```

| charslst ',' CHARCONS
= { pslstnxt($3) ; } ;
trmlst:
| tlstpars
= { $$ = pslstnxt(TERMLIST) ; } ;
tlstpars:
| tlstparam
= { pslstnew($1) ; }
| tlstpars ',' tlstparam
= { pslstnxt($3) ; } ;
tlstparam:
| ORG
= { $$ = pstreeno(ORG) ; }
| LOCATION
= { $$ = pstreeno(LOCATION) ; }
| CLEAR
= { $$ = pstreeno(CLEAR) ; }
| TELE
= { $$ = pstreeno(TELE) ; }
| COMP
= { $$ = pstreeno(COMP) ; }
| APP
= { $$ = pstreeno(APP) ; }
| FILE
= { $$ = pstreeno(FILE) ; } ;
clearance:
| TUNCL
= { $$ = pstreeno(TUNCL) ; }
| TTSSI
= { $$ = pstreeno(TTSSI) ; }
| TTSTK
= { $$ = pstreeno(TTSTK) ; }
| TTSST
= { $$ = pstreeno(TTSST) ; }
| TTSON
= { $$ = pstreeno(TTSON) ; }
| TSSI
= { $$ = pstreeno(TSSI) ; }
| TSTK
= { $$ = pstreeno(TSTK) ; }
| TSST
= { $$ = pstreeno(TSST) ; }
| TSON
= { $$ = pstreeno(TSON) ; }
| TCSI
= { $$ = pstreeno(TCSI) ; }
| TCON
= { $$ = pstreeno(TCON) ; } ;

```

APPENDIX B

TAS COMMAND SUMMARY AND SYNTAX

This appendix presents a brief description and syntax of each TAS command. Preceding the commands are basic definitions which are used in the command syntax.

Basic Definitions:

<application>	= < batch-application> < interactive-application >
<batch-application >	= DIAOLS RYETIP ISSPIC
<interactive-application>	= SOLIS
<terminal-ID>	= < integer >
<filename >	= one to thirteen character sequence of < letter> , <digit> or '.'
<jobid >	= ten-character JOBID returned for an entered query
<job-file-specifier >	= < filename> < jobid >
<user-ID>	= eight-character SNI
<date-time-group>	= <mo><day><hour><min> <mo><day><hour><min><year>
<mo>	= two digit month (01-12)
<day>	= two digit day (01-31)
<hour >	= two digit hour (00-23)
<min>	= two digit minute (00-59)
<year>	= two digit year (00-99)

Commands

AUTH display a list of authorized files for a terminal or for a user for one or all applications.

<auth-command> = AUTH | AUTH <auth-clause>

<auth-clause> = <application> |
 <terminal-ID> |
 <terminal-ID> <application>

Batch Query Processor (BQP)

process a new or canned batch query to a specified application.

<bpq-command> = <batch-application> |
 <batch-application> <filename>

CANCEL cancel one or all pending batch queries for a user.

<cancel-command> = CANCEL <cancel-clause>
 <cancel-clause> = <jobid> | ALL

CLOSE display batch query statuses and purgeable queries for a user.

<close-command> = CLOSE

DATEIS display or set the current date.

<dateis-command> = DATEIS | DATEIS <date-time-group>

DISPLAY display the results of an answered batch query.

<display-command> = DISPLAY <display-clause>
 <display-clause> = <jobid> | <user-ID> <jobid>

ECHO verify the correct operation of a terminal.
 <echo-command> = ECHO

EDIT create and/or edit a text file.
 <edit-command> = EDIT | EDIT <filename >

FILIST list all text files and batch jobs belonging to one or all TAS users.
 <filist-command> = FILIST | FILIST <user-ID >

HELP display a list of commands and a brief description of each or
 display a complete description of a specified command.
 <help-command> = HELP | HELP <command >

HOLD retain one or all batch queries/responses.
 <hold-command> = HOLD <hold-clause >
 <hold-clause> = <jobid> | ALL

Interactive System Processor (ISP)

 connect user to the specified interactive application.
 <isp-command> = <interactive-application >

JOBID display or set the next available TAS JOBID number.
 <jobid-command> = JOBID | JOBID <jobid-num >
 <jobid-num> = sequence of five digits

JOBSTAT display the status of one or all batch queries for a user.
 <jobstat-command> = JOBSTAT | JOBSTAT <jobid >

LIST display the text filenames belonging to a user.
 <list-command> = LIST

LOGOFF log the user off of TAS and purge non-retained, delivered queries/ responses.

<logoff-command> = LOGOFF

LOGTAPE mount a logging tape for the COINI logging records.

<logtape-command> = LOGTAPE

MSG display messages or send a new or canned message to a specified user.

<msg-command> = MSG | MSG <msg-clause>

<msg-clause> = <user-ID> | <user-ID> <filename>

PRINT print the results of an answered batch query.

<print-command> = PRINT <print-clause>

<print-clause> = <jobid> | <user-ID> <jobid>

PROJECT specify a project on which the user will work.

<project-command> = PROJECT | PROJECT <project-name>

<project-name> = sequence of one to ten letters

PURGE delete a user text file or a delivered query/response.

<purge-command> = PURGE <job-file-specifier>

RECLAIM reclaim TAS file system space.

<reclaim-command> = RECLAIM

RECOVER recover from a TAS system crash.

<recover-command> = RECOVER

REMOVE remove a specified user's text file or batch query job.

 <remove-command> = REMOVE <remove-clause >
 <remove-clause> = <user-ID> <job-file-specifier>

RESEND resend a batch query which is either pending or delivered.

 <resend-command> = RESEND <jobid >

RESUME resume a suspended interactive session.

 <resume-command> = RESUME

SAVE save a batch query as a user text file.

 <save-command> = SAVE <jobid> <filename >

SHOW display the contents of a user's text file or batch query.

 <show-command> = SHOW <show-clause >
 <show-clause> = <job-file-specifier>|
 <user-ID> <job-file-specifier >

SRVC send a service message to one or all batch applications.

 <srvc-command> = SRVC <srvc-clause >
 <srvc-clause> = <status-clause>|
 <print-clause >|
 <trace-clause>

 <status-clause> = <function> <application-specifier >
 <function> = CHECK | READY | BREAK

 <application-
 specifier> = <batch-application > | ALL

 <print-clause> = PRINT <application-specifier >|
 PRINT <application-specifier > <filename >

<trace-clause> = TRACE <application-specifier> |
TRACE <batch-application> <jobid>

SPURGE purge pending batch jobs following a reload from a backup magnetic tape or disk.

<spurge-command> = SPURGE

STATUS display the network status of one or all applications.

<status-command> = STATUS | STATUS application |
STATUS TAS

SYSRES obtain the percentage of TAS file system resources which are currently used.

<sysres-command> = SYSRES

SYSTAT obtain the status of pending TAS batch jobs for one or all batch applications.

<sysstat-command> = SYSTAT | SYSTAT <batch-application>

TALEND save or delete the TAS activity log.

<talend-command> = TALEND | TALEND <filename>

TALIST list TAS activity from the current TAS activity log (TAL) file or from a saved TAL file

<talist-command> = TALIST | TALIST <talist-clause>

<talist-clause> = <event-specifier> |
<time-range> |
<event-specifier> <time-range>

<event-specifier> = USER <user-ID> |
JOB <jobid> | LOG | VIO

<time-range> = <start-clause> <end-clause>

<start-clause> = START | <date-time-group>
<end-clause> = END | <date-time-group>

TASDOWN bring TAS down in a controlled fashion.

<tasdown-command>= TASDOWN <tasdown-clause>
<tasdown-clause> = <delay-clause> | <time-clause>
<delay-clause> = +<minutes> | - <minutes>
<time-clause> = <hour> : <min>
<minutes> = <integer>

TASDUMP back-up the TAS file system onto magnetic tape.

<tasdump-command>= TASDUMP

TASLOAD reload the TAS file system from magnetic tape.

<tasload-command> = TASLOAD

TRACE send a SRVC TRACE message to a batch application to see if a batch query job is still active.

<trace-command> = TRACE <jobid>

TSUM generate a TAS traffic summary (TSUM) report.

<tsum-command> = TSUM

USTAT determine the status of all or logged on TAS users.

<ustat-command> = USTAT | USTAT ALL

APPENDIX C

TAS SYSTEM PROCESS INTERFACES

This appendix supplies all the necessary interface information required to communicate with the system processes residing under TAS. In this context, system processes are those processes which are fundamental in providing application processes (i.e., those processes dealing directly with user capabilities) a direct tie-in to network transmissions, system logging, and security access authorization. Four basic system interfaces exist in TAS:

- a. Batch Network Communications
- b. Interactive Network Communications
- c. TAS Logging
- d. Access Authorization

These four system interfaces are discussed below.

Batch Network Communications

Although the Intelligent Network Interface (INI) process and its daemons are responsible for interfacing directly with the Network Control Program (NCP) for the emulation of COINS I message protocols through the ARPANET-like COINS II Network, the Batch Query and Response Dispatcher (BQRD) is the process which application processes (e.g., Batch Query Processors) actually call for the transmission of batch messages. BQRD receives a set of function codes and a variable set of parameters, depending on the function. BQRD is initiated through a standard fork/execute or execute sequence.

Before specifying the BQRD function/parameter set, a short description of a TAS job identifier (JOBID) is required. Each TAS unique JOBID will be created by the process which requires it. The JOBID will be ten characters in length with the first character assigned to an application from a master list.

The use of an alphabetic character in this position allows up to 26 unique applications. Currently, the following assignments have been made.

A - ARPA Data Access and Presentation Terminal (ADAPT)

D - DIAOLS Batch Query Processor (BQP)

I - ISSPIC BQP

R - RYETIP BQP

S - Internal TAS System Processes

The second, third, and fourth characters are somewhat arbitrary. In general, jobs generated by a specific user usually have that user's ID inserted in the JOBID. Arbitrary assignments to TAS internal system processes are as follows.

LOG - TAS Logging Processes (TLP)

BQD - Batch Query and Response Dispatcher (BQRD)

INI - Intelligent Network Interface (INI)

SOL - Interactive Query Interface (IQI) for SOLIS sessions

SRV - SRVC

The remaining six characters are generated via a call to the TAS global function, GFJOBID. This function appends six characters to a JOBID, with

the first of the six being a letter or digit which uniquely defines the current TAS host. The remaining five characters form an integer which is continually being incremented. It starts with 00000, is incremented to 99999, and re-starts again with 00000.

Examples of use:

BQP JOBID

bxxxzyzzzzz

b - BQP identifier, 'D', 'I', 'R'.

xxx - TAS user-ID.

y - Unique TAS host identifier.

zzzzz - Unique transaction number.

e. g. R005000232,

D057000233,

I021B00472

Internal System JOBID

cxxxzyzzzzz

c - Internal System, 'S'.

xxx - System Process Identifier: BQD, INI, SOL, SRV, LOG.

y - Unique TAS host identifier.

zzzzz - Unique Transaction Number.

e. g. SBQD001025,

SL0GB00239,

SSRV001026

BQRD Calling Sequence:

Fork/Execute with a wait on status (to clean UNIX process table) or Execute.

BQRD Input Parameters:

Function 1 - INTG

Input Parameters:

1. Function Code = 1
2. Host ID
3. JOBID
4. Pathname of query file
5. Pathname of INTG response processor

Note: query filenames are passed via GQFIL in the user's login directory.

Function 2 - SRVC (PRINT)

Input Parameters:

1. Function Code = 2
2. Host ID
3. JOBID
4. Pathname of print data file

Function 3 - SRVC (TRACE)

Input Parameters:

1. Function Code = 3
2. Host ID

3. JOBID (optional)
4. Pathname of file containing JOBID to trace (only if JOBID is given)
5. Pathname of TRACE response processor (only if JOBID is given)

Function 4 - SRVC (CHECK)

Input Parameters:

1. Function Code = 4
2. Host ID
3. JOBID
4. ASCII zero
5. Pathname of SRVC response processor

Function 5 - SRVC (READY)

Input Parameters:

1. Function Code = 5
2. Host ID
3. JOBID

Function 6 - SRVC (BREAK)

Input Parameters:

1. Function Code = 6
2. Host ID
3. JOBID

Function 7 - Cancel

Input Parameters:

1. Function Code = 7
2. Host ID
3. JOBID

Note: The remaining BQRD calls are performed by INI using an EXECUTE.

Function 8 - RLSE

Input Parameters:

1. Function Code = 8
2. Host ID
3. JOBID
4. Network ID

Function 9 - ANSR

Input Parameters:

1. Function Code = 9
2. Host ID
3. JOBID
4. Pathname of response file
5. Classification of Answer
6. Network ID
7. Segment count of ANSR

Function 10 - ABRT (FAULT)

Input Parameters:

1. Function Code = 10
2. Host ID
3. JOBID
4. Pathname of response file
5. Classification (not usable)
6. Network ID
7. Segment count of ABRT

Function 11 - ABRT (CAUSE)

Input Parameters:

1. Function Code = 11
2. Host ID
3. JOBID
4. Pathname of response file
5. Classification (not usable)
6. Network ID
7. Segment count of ABRT

Function 12 - Message Logged Out

Input Parameters:

1. Function Code = 12
2. Host ID
3. JOBID
4. Segment count of original message

Function 14 - SRVC (PRINT) RCPT

Input Parameters:

1. Function Code = 14
2. Host ID
3. JOBID

Function 15 - SRVC (CHECK) RCPT

Input Parameters:

1. Function Code = 15
2. Host ID
3. JOBID

Function 16 - SRVC (READY) RCPT

Input Parameters:

1. Function Code = 16
2. Host ID
3. JOBID

Function 17 - SRVC (BREAK) RCPT

Input Parameters:

1. Function Code = 17
2. Host ID
3. JOBID

Function 18 - SRVC (TRACK)

Input Parameters:

1. Function Code = 18
2. Host ID
3. JOBID being traced
4. Ten character Response to TRACE

Function 19 - SRVC (PRINT) received

Input Parameters:

1. Function Code = 19
2. Host ID
3. ASCII zero
4. Pathname of print data file

Function 20 - SRVC (READY) received

Input Parameters:

1. Function Code = 20
2. Host ID

Function 21 - SRVC (BREAK) received

Input Parameters:

1. Function Code = 21
2. Host ID

Function 22 - SRVC (ALARM received

Input Parameters:

1. Function Code = 22
2. Host ID

Function 23 - Network Read Error (Connection Closed)

Input Parameters:

1. Function Code = 23
2. Host ID

BQRD Output Sequences:

Function 1 - INTG

BQRD calls INI with the following parameters.

1. Function Code = 1
2. Host ID

3. JOBID
4. Pathname of query file

Function 2 - SRVC (PRINT)

BQRD calls INI with the following parameters.

1. Function Code = 2
2. Host ID
3. JOBID
4. Pathname of print data file

Function 3 - SRVC (TRACE)

BQRD calls INI with the following parameters.

1. Function Code = 3
2. Host ID
3. JOBID
4. Pathname of data file containing JOBID to trace

Function 4 - SRVC (CHECK)

BQRD calls INI with the following parameters.

1. Function Code = 4
2. Host ID
3. JOBID

Function 5 - SRVC (READY)

BQRD calls INI with the following parameters.

1. Function Code = 5
2. Host ID
3. JOBID

Function 6 - SRVC (BREAK)

BQRD calls INI with the following parameters.

1. Function Code = 6
2. Host ID
3. JOBID

Function 7 - Cancel

Cancel causes no output sequences.

Function 8 - RLSE

RLSE causes no output sequences.

Function 9 - ANSR

BQRD calls the process whose pathname is stored with the JOBID with the following parameters.

1. Function Code = 0
2. JOBID
3. Pathname of response file
4. Classification of Answer

Function 10 - ABRT (FAULT)

BQRD calls the process whose pathname is stored with the JOBID with the following parameters.

1. Function Code = 1
2. JOBID
3. Pathname of response file

Function 11 - ABRT (CAUSE)

BQRD calls the process whose pathname is stored with the JOBID with the following parameters.

1. Function Code = 2
2. JOBID
3. Pathname of response file

Function 12 - Message Logged Out

If the logged out message is an INTG, then BQRD calls the process whose pathname is stored with the JOBID with the following parameters.

1. Function Code = 3
2. JOBID

If the logged out message is a SRVC (CHECK), then LQRD calls the process whose pathname is stored with the JOBID with the following parameters.

1. Function Code = 16
2. Host ID

If the logged out message is a SRVC (TRACE) not generated by automatic traces, then BQRD calls the process whose pathname is stored with the JOBID with the following parameters.

1. Function Code = 15
2. JOBID being traced

For all logged out messages, BQRD calls the SRVC Response Processor (SRESP) with the following parameters.

1. Function Code = 12 Message logged out
2. Host ID
3. Job Type (1 = INTG, 2 = SRVC (PRINT), etc.)

Function 13 - INTG Lost

This is an internal BQRD code generated when a user's SRVC (TRACE) indicates No Record in the host. BQRD calls the process whose pathname is stored with the INTG with the following parameters.

1. Function Code = 4
2. JOBID

Function 14 - SRVC (PRINT) RCPT

This function causes no output sequences.

Function 15 - SRVC (CHECK) RCPT

BQRD calls the process whose pathname is stored with the JOBID with the following parameters.

1. Function Code = 1
2. Host ID

Function 16 - SRVC (READY) RCPT

SRVC (READY) RCPT causes no output sequences.

Function 17 - SRVC (BREAK) RCPT

BQRD calls the process responsible for processing SRVC input messages with the following parameters.

1. Function Code = 21
2. Host ID

A SRVC (BREAK) RCPT is processed exactly like a SRVC (BREAK) received.

Function 18 - SRVC (TRACK)

BQRD calls the process whose pathname is stored with the JOBID with the following parameters.

1. Function Code = 4
2. JOBID being traced
3. Ten character response to TRACE

Function 19 - SRVC (PRINT) received

BQRD calls the process responsible for processing SRVC input messages with the following parameters.

1. Function Code = 19
2. Host ID
3. Pathname of print data file

Function 20 - SRVC (READY) received

BQRD calls the process responsible for processing SRVC input messages with the following parameters.

1. Function Code = 20
2. Host ID

Depending on what kinds of jobs are pending for the Host ID, BQRD may also output INTG, SRVC (PRINT), or SRVC (TRACE) messages to INI.

Function 21 - SRVC (BREAK) received

BQRD calls the process responsible for processing SRVC input messages with the following parameters.

1. Function Code = 21
2. Host ID

Function 22 - SRVC (ALARM) received

BQRD calls the process responsible for processing SRVC input messages with the following parameters.

1. Function Code = 22
2. Host ID

Function 23 - Network Read Error (Connection Closed)

BQRD calls the process responsible for processing SRVC input messages with the following parameters.

1. Function Code = 23
2. Host ID

Interactive Network Communications

The Interactive Query Interface (IQI) process and its network read counterpart, IQIR, are responsible in TAS for all interactive communications through the COINS II Network. These processes perform the following functions:

- a. Establish an Initial Connection Protocol (ICP) with the designated interactive host.
- b. Provide direct interfaces for all network reading and writing to interactive hosts.
- c. Perform transparent user host logon sequences for the designated user and host.
- d. Perform all necessary network logging as required for interactive networking via the appropriate system logging processes.

IQI and IQIR processes interface directly with the Network Control Program (NCP), where they initiate the ICP via a UNIX open system call to the appropriate network host, and transfer data to and from the host through this connection to the application process.

The IQI process is invoked by a TAS application process when an interactive connection is required. IQI in turn invokes IQIR for network reading.

IQI Calling Sequence:

Fork/Execute with NO wait on status.

IQI Input Parameters:

1. Host ID
2. File descriptor of the read-half of the TO-HOST pipe

3. File descriptor of the write-half of the FROM-HOST pipe
4. Process-ID of calling process

Status Returned (via single word on FROM-HOST pipe):

- 0 = Connection/logon successful
- 1 = Host connection not successful
- 2 = User authentication (logon) not successful

IQI/Process Signal Communication:

Signal 7 (to IQI): Terminate interactive connection

Signal 7 (from IQI): Network error encountered, connection terminated

General Operation

In order to successfully initiate an interactive connection through IQI the calling process must establish two pipes, a TO-HOST pipe and a FROM-HOST pipe. When IQI is spawned it requires a host ID, the read half of the TO-HOST pipe, the write half of the FROM-HOST pipe, and the calling process' process ID for signal communication. The calling process must then wait for IQI to return a status by reading two characters from the read half of the FROM-HOST pipe. Assuming successful ICP and user logon, the calling process must then control the user-host interaction (which in most cases is host specific) by reading and writing over the proper ends of the two pipes. IQI and IQIR are in most cases completely independent of the actual host (except for the host specific user logon sequence), and appropriately, expect the calling process to control all operations. IQI reads data from the TO-HOST pipe and writes it out to the host via the network. IQIR reads data from the network (via the host) and writes it out to the FROM-HOST pipe.

If the calling process decides to terminate the interactive connection, it must notify IQI by sending a signal 7. If IQI or IQIR detect a network write/read error (almost always caused by a closed connection from the host) then IQI will send a signal 7 to the calling process and then exit. In either case the calling process should wait for IQI to properly exit.

TAS Logging

The TAS Logging Processes (TLPs) consist of fourteen system processes which are responsible for all TAS logging. Each TLP (TLP00-TLP13) is activated by the fork/execute mechanism with a wait on the execution status, or by an execute. The first argument to a particular TLP is an event code; the nature and number of the remaining arguments depend on the event.

TLP00:

Event: A user logs onto TAS

Input Parameters:

1. Event code (0)
2. User's logon sni
3. (Un)successful logon flag - 0 successful
1 unsuccessful

TLP01:

Event: A user logs off of TAS

Input Parameters:

1. Event code (1)

AD-A050 967

LOGICON INC SAN DIEGO CALIF

F/G 9/2

TERMINAL ACCESS SYSTEM (TAS) SYSTEM DESIGN SPECIFICATION.(U)

JAN 78 R M BARNHART, L R ERICKSON

N00014-76-C-0899

UNCLASSIFIED

76-C-0899-6

NL

8 OF 8

AD
A050 967



END
DATE
FILMED

4-78

DDC

TLP02:

Event: A user submits a batch query

Input Parameters:

1. Event code (2)
2. JOBID
3. Application ID

Note: query filenames are passed via GQFIL in the user's login directory.

TLP03:

Event: Batch query released to network

Input Parameters:

1. Event code (3)
2. JOBID
3. RCPT (not) received flag - 0 received
1 not received
4. Segment count of INTG

TLP04:

Event: RISE received from batch host

Input Parameters:

1. Event code (4)
2. JOBID
3. Network ID
4. Application ID

TLP05:

Event: Response to batch query received

Input Parameters:

1. Event code (5)
2. JOBID
3. Response type -

0	No reply
1	ANSR
2	ABORT CAUSE
3	ABORT HOST
4	ABORT ERROR
5	ABORT FAULT
4. Network Response ID
5. Response classification (if ANSR)
6. Segment count of reply (if ANSR)

TLP06:

Event: Non-existent JOBID received from network

Input Parameters:

1. Event code (6)
2. JOBID
3. Application ID

TLP07:

Event: (Un)successful batch query response display

Input Parameters:

1. Event code (7)
2. JOBID
3. Authorization code - 0 User/terminal authorized for viewing response.
 - 1 User not authorized
 - 2 Terminal not authorized
4. Response Classification (A - K)

TLP08:

Event: Batch query cancelled

Input Parameters:

1. Event code (8)
2. JOBID

TLP09:

Event: SRVC TRACE returns "non-existent JOBID"

Input Parameters:

1. Event code (9)
2. JOBID
3. Application ID

TLP10:

Event: (Un)successful interactive host logon

Input Parameters:

1. Event code (10)
2. JOBID
3. Application ID
4. Logon flag - 0 successful
1 unsuccessful

TLP11:

Event: Interactive session termination

1. Event code (11)
2. JOBID
3. Application ID
4. Termination code - 0 successful
1 no reply
2 timed-out
3 too many retransmissions
4 unable to send/receive from network
5 addressee (host) down
6 session aborted by addressee
5. Number of characters transmitted to host
6. Number of characters received from host

TLP12:

Event: Unauthorized application access attempt

Input Parameters:

1. Event code (12)
2. Application ID
3. Access code - 0 User/term authorized to access host
1 User not authorized
2 Terminal not authorized

TLP13:

Event: Unauthorized batch-host-file access attempt

Input Parameters:

1. Event code (13)
2. Application ID
3. Access code - 0 User/terminal authorized access to
all referenced batch-host-files
1 User not authorized
2 Terminal not authorized

Note: query filenames are passed via GQFIL in the user's login directory.

Access Authorization

The Access Authorization Process (AAP) provides complete security access authorization verification for all of TAS. It is a single system process which provides access authorization in three major areas:

1. Determine whether a user/project/terminal is authorized to access a specified file(s) within an application.

2. Determine whether a given response clearance exceeds either a user's or a terminal's clearance.
3. Determine whether a user/project/terminal is authorized to access a specified application.

AAP Input Parameters:

Function 1 - Determine if a user/project and terminal are authorized to access a set of specified files within an application.

Calling Sequence:

Fork/Execute, Wait on status.

Input Parameters:

1. Function code = 1
2. Application identifier.

Note:

The filename(s)/access mode(s) (R for retrieve and U for update) for a given application are stored in GQFIL (batch query filenames for authorization check). For each file, AAP stores one of the following authorization status codes:

- 0 = access authorized for both user/project and terminal.
- 1 = access not authorized for user/project.
- 2 = access not authorized for terminal.

Status Returned (Summary of the individual files' authorization statuses):

- 0 = access authorized for both user/project and terminal.
- 1 = access not authorized for user/project.
- 2 = access not authorized for terminal.
- 3 = access not authorized for either the user/project or terminal.

Function 2 - Determine if response clearance exceeds either user or terminal clearance.

Calling Sequence:

Fork/Execute, Wait on status.

Input Parameters:

- 1. Function code = 2.
- 2. JOBID.
- 3. Response clearance.

Status Returned:

- 0 = clearance of user/terminal exceeds response
- 1 = clearance of response exceeds user
- 2 = clearance of response exceeds terminal

Function 3 - Determine if a user/project and terminal are authorized to access a specified application.

Calling Sequence:

Fork/Execute, Wait on status.

Input Parameters:

1. Function code = 3.
2. Application identifier.
3. Application mode of operation:
 - 0 = batch application
 - 1 = interactive application.

Status Returned:

- 0 = access authorized for both user/project and terminal.
- 1 = access not authorized for user/project.
- 2 = access not authorized for terminal.

DISTRIBUTION LIST

Defense Documentation Center Cameron Station Alexandria, VA 22314	12 copies
Office of Naval Research Information Systems Program Code 437 Arlington, VA 22217	2 copies
Office of Naval Research Code 102IP Arlington, VA 22217	6 copies
Office of Naval Research Code 200 Arlington, VA 22217	1 copy
Office of Naval Research Code 455 Arlington, VA 22217	1 copy
Office of Naval Research Code 458 Arlington, VA 22217	1 copy
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, MA 02210	1 copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, IL 60605	1 copy
Office of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, CA 91106	1 copy

New York Area Office 715 Broadway - 5th Floor New York, NY 10003	1 copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D.C. 20375	6 copies
Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D.C. 20380	1 copy
Naval Electronics Laboratory Center Advanced Software Technology Division Code 5200 San Diego, CA 92152	1 copy
Mr. E. H. Gleissner Naval Ship Research & Development Center Computation and Mathematics Department Bethesda, MD 20084	1 copy
Captain Grace M. Hopper NAICOM/MIS Planning Branch (OP-916D) Office of Chief of Naval Operations Washington, D.C. 20350	1 copy
Mr. Kin B. Thompson Technical Director Information Systems Division (OP-91T) Office of Chief of Naval Operations Washington, D.C. 20305	1 copy
Advanced Research Projects Agency Information Processing Techniques 1400 Wilson Boulevard Arlington, VA 22209	1 copy